

**A SOFTWARE BASED APCO PROJECT 25 DATA TRANSMISSION BASE  
STATION FOR LOCAL POLICE HEADQUARTERS**

BY

ERIC RENÉ RAMSEY

B.S., University of New Hampshire, 2005

THESIS

Submitted to the University of New Hampshire  
in Partial Fulfillment of  
the Requirements for the Degree of

Master of Science  
in  
Electrical Engineering

September, 2007

This thesis has been examined and approved.

---

Dr. W. Thomas Miller, III  
Thesis Director  
Professor of Electrical Engineering

---

Dr. Andrew L. Kun  
Associate Professor of  
Electrical Engineering

---

Dr. William H. Lenharth  
Research Associate Professor of  
Electrical Engineering

---

Date

## **DEDICATION**

*For my beautiful Wife, Colleen.  
Thank you for your Love and Support*

## **ACKNOWLEDGEMENTS**

I would like to take this opportunity thank all of the people who have guided and supported me during the Thesis processes:

Dr. Lenharth and Dr. Kun, for their mentoring and friendship.

Dr. Miller, my thesis advisor, for sharing his time and knowledge.

Scott, Coleen, and J.R., for their constant support.

My Mother and Father, who offered their Love, support, and excellent advise.

Last but not least, I would like to thank my loving and supporting wife Colleen, for sacrificing her nights and weekends so that I could work and for always listening to my problems.

## TABLE OF CONTENTS

|   |      |
|---|------|
| DEDICATION .....  | iii  |
| ACKNOWLEDGEMENTS.....                                   | iv   |
| LIST OF TABLES .....                                    | vii  |
| LIST OF FIGURES .....                                   | viii |
| LIST OF ACRONYMS .....                                  | x    |
| ABSTRACT .....  | xi   |
| INTRODUCTION .....                                      | 1    |
| I. APCO PROJECT 25 COMMON AIR INTERFACE .....           | 4    |
| Encoding and Transmission of Digital Data Packets ..... | 8    |
| II. THE TXRX APPLICATION.....                           | 12   |
| Project 25 Packet Conversion.....                       | 12   |
| Header Block .....                                      | 13   |
| Data Error Correction .....                             | 20   |
| Network Identifier .....                                | 23   |
| Frame Synchronization Word.....                         | 27   |
| Status Symbols .....                                    | 27   |
| Channel Initialization .....                            | 28   |
| Dibits to Frequencies .....                             | 28   |
| Nyquist Raised Cosine Filter.....                       | 29   |
| Pre-Emphasis Filter .....                               | 32   |
| Push-to-Talk Control .....                              | 35   |

|   |    |
|---|----|
| Sound Card Control .....                              | 36 |
| III. TESTING.....                                     | 37 |
| Testing Configuration.....                            | 37 |
| Software Testing.....                                 | 38 |
| Sound Card Testing.....                               | 41 |
| Analog Radio Modulation Testing Using A VSA .....     | 42 |
| Motorola Digital Radio Demodulation .....             | 48 |
| End-to-End IP Testing.....                            | 48 |
| IV. CONCLUSIONS.....                                  | 52 |
| Future Improvements.....                              | 54 |
| APPENDIX A – Motorola XTL500 Code Plug Settings ..... | 57 |
| APPENDIX B – Filter Design and Coefficients .....     | 62 |
| Nyquist Raised Cosine Filter.....                     | 62 |
| Pre-Emphasis Filter .....                             | 64 |
| APPENDIX C – IP Reflector in Project54 .....          | 66 |
| APPENDIX D – VSA Settings.....                        | 69 |
| APPENDIX E – Registry Parameters .....                | 70 |
| APPENDIX F – Custom Radio Interface Cable .....       | 72 |

## LIST OF TABLES

|   |    |
|---|----|
| Table I.1- Project 25 Frequency Values .....                      | 9  |
| Table II.1 - SAP Identifier Values .....                          | 14 |
| Table II.2 - Example Header Block .....                           | 17 |
| Table II.3- Rate 1/2 Trellis Encoder State Transition Table ..... | 21 |
| Table II.4 - Constellation to Dibit Pair Mapping .....            | 21 |
| Table II.5 - Interleave Table .....                               | 22 |
| Table II.6 - Network Identifier Information .....                 | 23 |
| Table II.7 - Data Unit Identifier Values .....                    | 24 |
| Table II.8 – BCH Code Generator Matrix .....                      | 26 |
| Table II.9 - True Network Identifier .....                        | 26 |
| Table II.10 - Frame Sync Word Sequence .....                      | 27 |
| Table II.11 - Status Symbol Codes .....                           | 27 |
| Table B.1 - Normalized Smoothing Filter Coefficients.....         | 63 |
| Table B.2 - Shaping Filter Coefficient Values .....               | 64 |

## LIST OF FIGURES

|   |    |
|---|----|
| Figure 1 - Overview of the Cruiser and Project 25 Analog Base Station.....            | 2  |
| Figure I.1 - Common Air Interface in a Radio System.....                              | 4  |
| Figure I.2 - Voice Versus Data Packets .....  | 5  |
| Figure I.3 - Commercial Voice and Data Base Station.....                              | 6  |
| Figure I.4 - Designed Voice and Data Base Station.....                                | 7  |
| Figure I.5 - IP Packet to CAI Data Packet Conversion.....                             | 8  |
| Figure I.6 - Project 25 Data Radio System Architecture.....                           | 9  |
| Figure II.1 - Top Level Flow of TxRx Application .....                                | 12 |
| Figure II.2 - Decomposition of Data Message into Short Blocks .....                   | 13 |
| Figure II.3 - Unconfirmed Data Packet Header Format.....                              | 13 |
| Figure II.4 - Unconfirmed Data Block .....  | 18 |
| Figure II.5 - Rate 1/2 Trellis Encoder Overview .....                                 | 20 |
| Figure II.6 - Trellis Encoder Block Diagram .....                                     | 20 |
| Figure II.7 - Packet Data Unit.....   | 28 |
| Figure II.8 - Actual Smoothing Filter Frequency Response.....                         | 29 |
| Figure II.9 - Impulse Response of the Smoothing Filter .....                          | 30 |
| Figure II.10 - Actual and Designed Magnitude Response.....                            | 31 |
| Figure II.11 - Example of Input and Output of the Smoothing Filter .....              | 32 |
| Figure II.12 - Running Average Filter Frequency Response .....                        | 33 |
| Figure II.13 - Ideal Pre-Emphasis Filter .....  | 34 |
| Figure II.14 - Magnitude Response of the Actual and Designed Pre-Emphasis Filter .... | 35 |
| Figure III.1 - Testing Configuration.....   | 37 |
| Figure III.2 - Output and Input to the Smoothing Filter (Vertical Axis in Hz).....    | 39 |

|  |    |
|--|----|
| Figure III.3 - Output and Input to the Pre-Emphasis Filter (Vertical Axis in Hz) ..... | 40 |
| Figure III.4 - Oscilloscope Reading of Test Signal .....                               | 41 |
| Figure III.5- Oscilloscope reading of the Frame Sync. ....                             | 42 |
| Figure III.6 - Frequency Spectrum using VSA.....                                       | 43 |
| Figure III.7 - New Test Sequence .....   | 44 |
| Figure III.8 – Female DB-25 Connector on Motorola Astro .....                          | 45 |
| Figure III.9 - Ideal and Distorted Radio Transmissions .....                           | 45 |
| Figure III.10 - Female DB-25 Connector on Kenwood TK-7180 .....                        | 46 |
| Figure III.11 - APCO 25 Demodulation (Time Measure) using VSA .....                    | 47 |
| Figure III.12 - APCO 25 Demodulation (Symbols/Errors) using VSA.....                   | 47 |
| Figure III.13 - Transmit Side of P25Messenger .....                                    | 50 |
| Figure III.14 - Receive Side of P25Messenger.....                                      | 50 |
| Figure IV.1 - Overview of the Project 25 Analog Base Station.....                      | 52 |
| Figure B.1 - Normalized Smoothing Filter Coefficients .....                            | 63 |
| Figure B.2 - Shaping Filter Coefficients .....   | 65 |
| Figure C.1 - The IP Reflector Architecture .....                                       | 67 |
| Figure C.2 - IP Packet from IP Network Application to "IP Reflector" Driver.....       | 67 |
| Figure C.3 - IP Packet from "IP Reflector" Driver to IP Reflector Service .....        | 67 |
| Figure C.4 - IP Packet from IP Reflector Service to Custom Data Modem Software .....   | 68 |
| Figure F.1 - Custom Interface Cable .....  | 72 |

## LIST OF ACRONYMS

|      |   |
|------|---|
| APCO | Association of Public-Safety Communications Officials |
| BCH  | Bose, Ray-Chaudhuri, Hocquenghem                      |
| CAI  | Common Air Interface                                  |
| CRC  | Cyclic Redundancy Check                               |
| DTR  | Data Terminal Ready                                   |
| FIR  | Finite Impulse Response                               |
| FS   | Frame Synchronization                                 |
| FSK  | Frequency Shift Keying                                |
| I/O  | Input and Output                                      |
| IP   | Internetwork Protocol                                 |
| MFID | Manufacturer Identification                           |
| MOD  | Modulo 2  |
| NAC  | Network Access Code                                   |
| NID  | Network Identification                                |
| PTT  | Push-to-Talk  |
| SAP  | Service Access Point                                  |
| TxRx | Transmit and Receive                                  |
| XOR  | Exclusive OR  |

## **ABSTRACT**

### **A SOFTWARE BASED APCO PROJECT 25 DATA TRANSMISSION BASE STATION FOR LOCAL POLICE HEADQUARTERS**

by

Eric René Ramsey  
University of New Hampshire, September, 2007

Project54 is a research and development effort to integrate embedded and wireless technologies into both NH State Police and local police cruisers. One of the features of the Project54 system, is the ability to query motor vehicle data. This is done through a Motorola APCO Project 25 digital voice/data radio in the cruiser and a digital voice/data base station at headquarters. An APCO Project 25 base station is a central hub for the transmission and reception of digital voice/data packets. The local police departments have the requisite radio infrastructure, but do not have the budget to purchase and support a Motorola digital voice/data base station.

The main goal of this thesis was to develop a low-cost data transmission base station to transmit data from a police headquarters to a cruiser. The base station developed uses an analog FM transmitter interfaced to a standard desktop computer, which implements the APCO Project 25 compliant digital data modulation.

## INTRODUCTION

Project54 is a collaborative research and development effort between the University of New Hampshire and the New Hampshire Department of Safety and is supported by the U.S. Department of Justice. The Project54 system integrates cruiser-based electronic systems to offer advanced support to New Hampshire State Troopers and to Local Police Officers (1). One of the many electronic components integrated in the system is the Motorola Astro digital radio. This specific model can be found in both the NH State Police cruisers and in all of the local department cruisers. However, the full voice and data capabilities of the digital radio are only utilized by the New Hampshire State Police. The complex digital radio network infrastructure being used by the New Hampshire State Police includes radios, antennae, repeaters, and base stations. The combination of all the components is too costly for local departments. The local departments do not need some of the more complex features the Motorola base station provides. It is for these two reasons, cost and complexity, that local departments use the voice functionality, but not the data functionality of their radio system. The main goal of this thesis is to design, develop, and prepare for implementation, a cost effective way for local departments to transmit digital data from headquarters to cruisers. The result is the Project 25 Analog Base Station.

The equipment needed to communicate between headquarters and a cruiser is shown below in Figure 1.

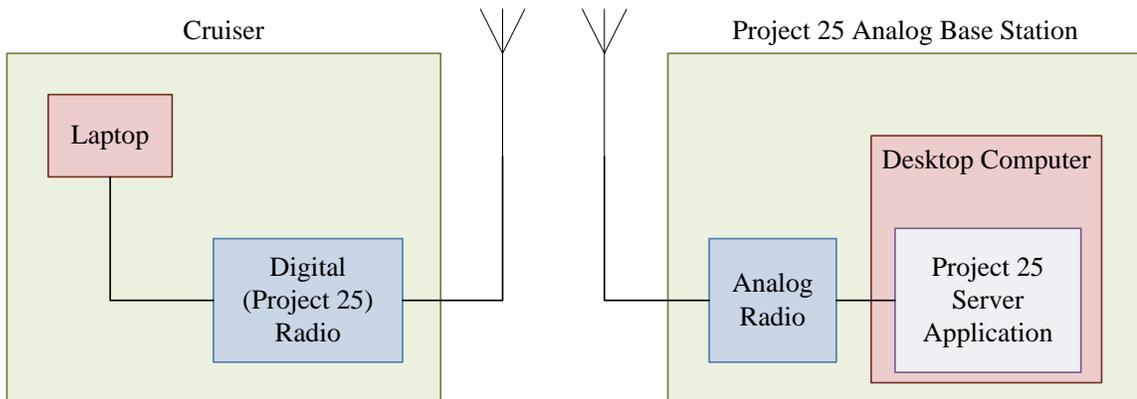


Figure 1 - Overview of the Cruiser and Project 25 Analog Base Station

The Project 25 analog base station will be considered the server while the cruiser will be considered the client. The server is composed of a computer and an analog radio. An Internet Protocol (IP) Network Application at headquarters sends out an IP packet bound for the laptop in the cruiser. This IP Packet is captured by software on the base station computer. It is then encoded and filtered using the APCO Project25 Common Air Interface (CAI) standard. The output of the application is a Project25 digital waveform modulation which is played through the computer's sound card. The computer controls and transmits sound through the analog radio. When triggered, the analog radio broadcasts the waveform on a predefined frequency.

Once the analog radio transmits the packet, the cruiser, or client, must receive and properly decode the message. The client's radio is a digital radio tuned to the same frequency as the analog radio. The difference is that the digital radio only receives transmissions with Project 25 digital modulations. The transmitted waveform is received on the digital radio and is demodulated, filtered

and decoded. The result is passed to the cruiser's in-car computer in the form of an IP packet, completing the task.

In the first chapter of this thesis, the Project 25 Common Air interface standard will be presented. During the discussion of the standard, a commercial voice and data base station will be compared to the designed voice and data base station. The chapter will also introduce the concept of encoding and transmission of digital data packets.

Chapter II will discuss the application which was designed to encode IP Packets as Project 25 Common Air Interface data packets. Some of the topics covered in the chapter include header and data block formation, data error correction, and base band filtering.

The third Chapter discusses the testing of the whole base station. This is broken into testing sections, where each section tests a different part or component of the designed base station. The testing chapter also discusses the successful results of the Project 25 analog base station.

Chapter IV discusses the conclusions drawn after the design, implementation and testing were completed. This chapter also provides an overview of the finished design and offers future improvements to the system.

## CHAPTER I

### APCO PROJECT 25 COMMON AIR INTERFACE

The APCO (Association of Public-Safety Communications Officials) Project 25 Systems and Standards Definition describes an open interface between mobile, portable, and base station radios. The objective of Project 25 is to allow effective and reliable intra-agency and inter-agency communications in a digital mode for both voice and data. Vendors who adhere to the standard will produce a system which is interoperable with other standardized equipment. Project 25 documents describe a number of open interfaces to a reference point, such that standardized equipment can communicate through the interfaces.

A key open interface within the Project 25 standard is the Common Air Interface. The Common Air Interface (CAI) focuses on how to transmit digital voice and data over a radio channel, as shown in Figure I.1.

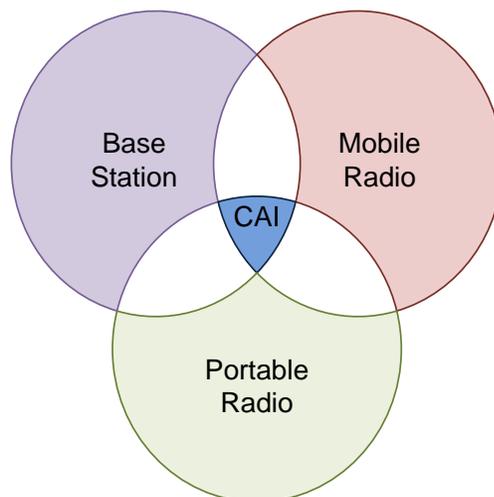


Figure I.1 - Common Air Interface in a Radio System

The rate at which the communication occurs over the Common Air Interface is 4800 symbols per second, where a symbol is composed of 2 bits, also known as a dibit. These dibit symbols can take on one of four values relating to four different frequencies. This style of modulation is called C4FM, or 4-level frequency modulation.

Both voice and data packets can be transmitted and received on the same digital channel, using the 4-level frequency modulation. However, to manage voice and data communications on the same channel, voice packets are granted priority over data packets. This is illustrated in Figure I.2. Voice packets take up the full bandwidth of the signal. Data packets can only be transmitted in the silence, between voice messages. The distinction between voice and data packets lies in the control bits present at the beginning of each Common Air Interface voice or data packet.



Figure I.2 - Voice Versus Data Packets

The reception and transmission of voice and data packets occurs at the base station. A commercial voice and data base station configuration is shown in Figure I.3.

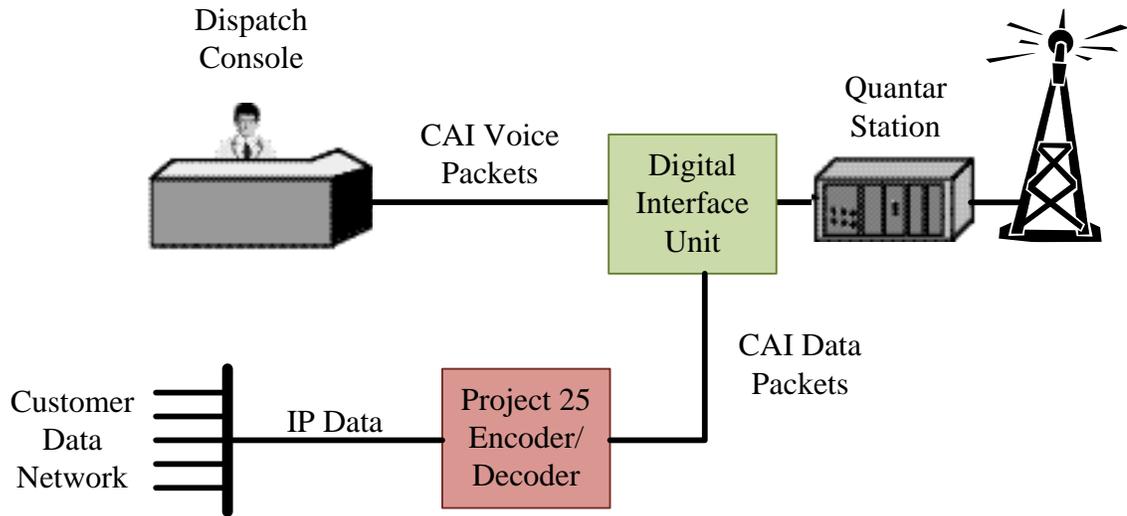


Figure I.3 - Commercial Voice and Data Base Station

With this setup, CAI transmissions are received by the Quantar Station. The demodulated digital packet is then fed to the Digital Interface Unit. The Digital Interface Unit reads the control bits at the beginning of the packet and steers voices packets to the dispatch console and data packets to the Project 25 Encoder/Decoder. The output at the dispatch console is an analog voice waveform that is played over an external speaker. To respond to the voice transmission, the dispatcher speaks into a microphone. The analog voice waveform is transformed into a CAI voice packet which is sent from the console, through the Digital Interface Unit, to the Quantar Station. At the Quantar Station, the voice packet is frequency modulated, then transmitted. The output of the Project 25 Encoder/Decoder is a standard IP data packet. For the transmission of a data packet, information is fed to the Project 25 Encoder/Decoder in the form of an IP data packet. The output is a CAI Data Packet which is sent to the Digital

Interface Unit. The DIU holds onto the data packet until there is a break in voice transmissions. Once there is silence, the data packet is passed to the Quantar Station for frequency modulation and is transmitted.

The voice and data base station designed in this thesis, see Figure I.4, uses the preexisting voice base station, but adds a custom base station for data transmission.

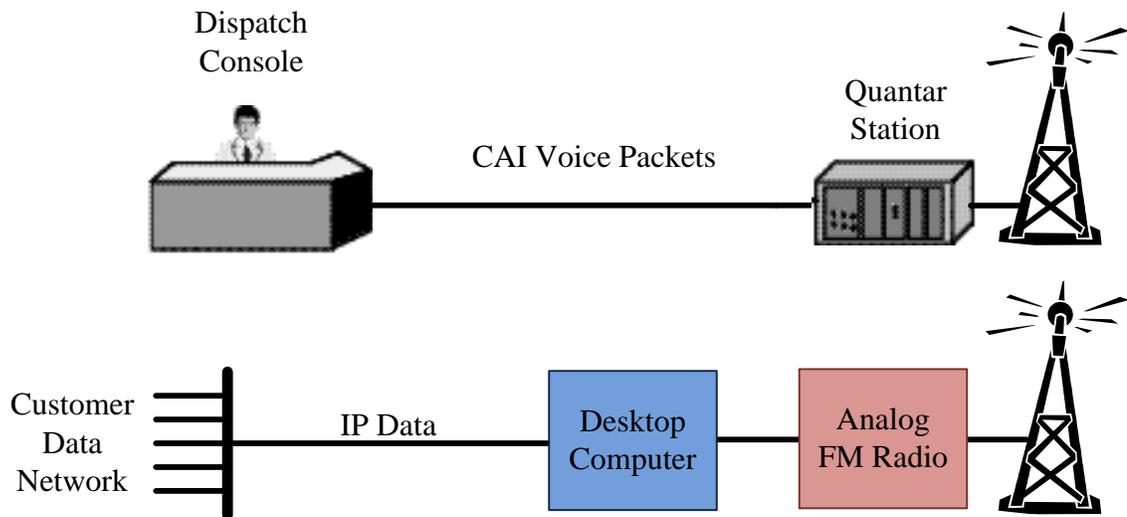


Figure I.4 - Designed Voice and Data Base Station

This setup receives voice messages at the Quantar Station. The CAI voice packets are passed directly to the dispatch console. The transmission of a voice packet occurs in the same manner as the commercial voice transmission, except it does not have to pass through the Digital Interface Unit. For the reception and transmission of data packets, an analog radio and a desktop computer (PC) are used. The analog radio receives both voice and data packet transmissions. The analog version of the digital packet is passed to the computer through the

microphone input. This signal is then decoded through software. Voice packets are ignored, while data packets are decoded into IP data packets. The IP data packets are then placed on the customer data network through the computers network card. For the transmission of data packets, IP data packets are received by the computer and encoded into CAI data packets. These CAI data packets are then passed through the speaker output of the computer to the analog radio where they are transmitted. The reception and decoding of data packets is outside the scope of this thesis and will be discussed in the future work section. The encoding and transmission of data packets is the main focus of this thesis.

### **Encoding and Transmission of Digital Data Packets**

Digital data messages are transferred over the Common Air interface using a packet technique which is discussed further in Chapter II.

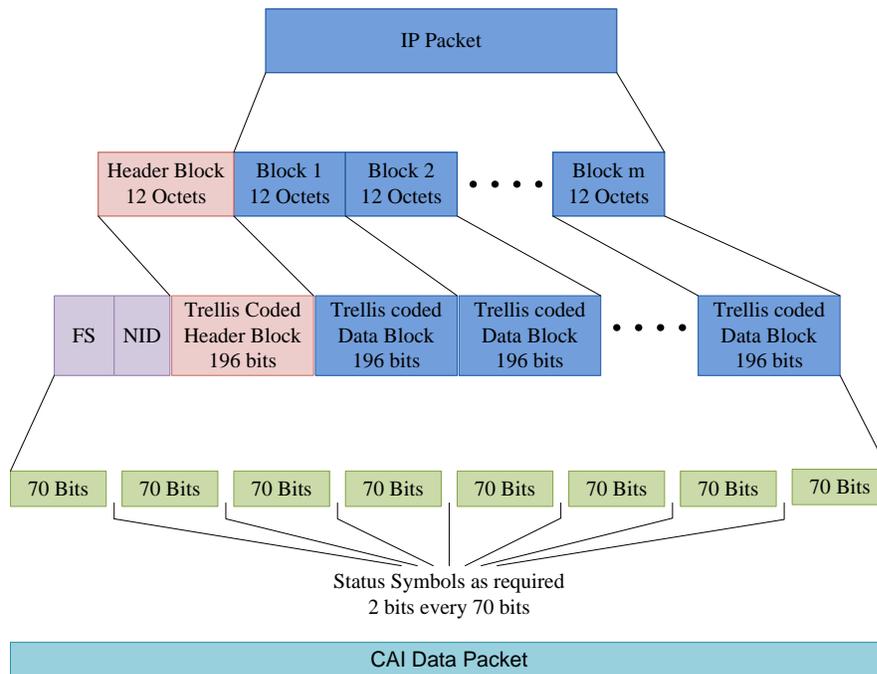


Figure I.5 - IP Packet to CAI Data Packet Conversion

The IP data packet, or data message, is first sent through Project 25 Encoding. See Figure I.5.

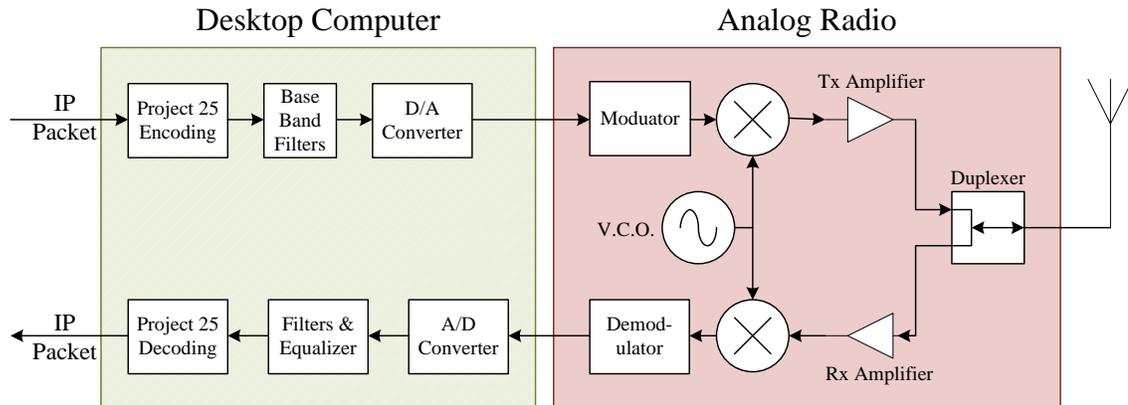


Figure I.6 - Project 25 Data Radio System Architecture

The Packet is segmented into blocks of 12 octets and a header block is added. Each block is protected by a 1/2 rate trellis code. Once encoded, the header and data blocks are placed together to form a sequence. A preamble, composed of a frame synchronization and network identifier, is then added to the encoded sequence. The result is then broken into segments of 35 symbols (70 bits), separated by 1 status symbol. Each symbol is composed of one dibit. This results in four different values related, through C4FM, to four different FM frequency deviations. The different values and their corresponding frequency deviations are listed in Table I.1.

Table I.1- Project 25 Frequency Values

| Symbol Value | Dibit | Frequency (Hz) |
|--------------|-------|----------------|
| 0            | 00    | 600            |
| 1            | 01    | 1800           |
| 2            | 10    | -600           |
| 3            | 11    | -1800          |

After the frequency deviations are up-sampled and filtered by a Nyquist raised cosine and pre-emphasis base band filters, the complete data packet is passed to a digital to analog (D/A) converter and transferred over the Common Air Interface through an analog radio (Figure I.6).

The Project 25 CAI standard is becoming popular in radio communications for both voice and data because of its interoperability between mobile radios and minimal usage of frequency bands. Many third party companies are designing APCO P25 Base Stations. One example of this is a company called Etherstack. Etherstack has a comprehensive range of APCO P25 compliant Base Station and Repeater options. These options are written in highly portable ANSI C/C++. If a third party radio company is building a base station or a repeater, the Project 25 standard code can be purchased from Etherstack and compiled onto the digital signal processing chip of the radio in use. (2)

Other work has been done by Vanu, Inc. Vanu, Inc., in partnership with General Dynamics Decision Systems, developed a handheld software radio prototype to address interoperability issues between analog and digital radios. The Vanu software radio supports two waveforms: half-duplex analog FM and digital APCO Project 25. The software defined radio is able to switch between the two half-duplex waveforms. This setup is advantageous to departments interested in voice interoperability, but it does not have any data capabilities. (3)

Project 25 CAI Voice packet conversion has even been brought down to the level of digital signal processing chips. The AMBE-2000 Vocoder Chip, produced by Digital Voice Systems, Inc., is an extremely flexible, high performance voice compression solution. It provides exceptional voice quality at rates as low as 2000 bits per second. This chip can be used for the encoding and compression of voice data. However, it cannot be used for the encoding of Project 25 CAI Data Packets. (4)

The main focus of Project 25 compatibility and integration has been in the development of voice transmission and reception. An inexpensive, simple, stand alone data transmission and reception base station has never before been developed or deployed.

## CHAPTER II

### THE TXRX APPLICATION

The TxRx application developed in this project is a central hub for data flow throughout the Project 25 server. It is a console application which receives an IP packet, encodes and modulates it using definitions set forth by the APCO Project 25 Standard, presses the push to talk (PTT) of the radio, and transmits the resulting data packet. A top level flow diagram of the TxRx application is shown in Figure II.1.

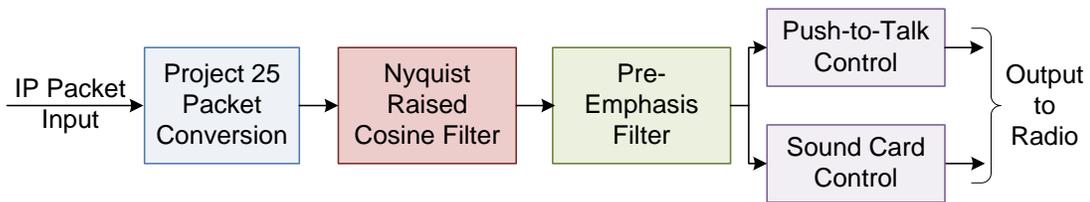


Figure II.1 - Top Level Flow of TxRx Application

#### **Project 25 Packet Conversion**

The input to the TxRx program is an IP Packet with length no longer than 512 octets. This IP Packet is received from the Project54 standard IP Reflector (Appendix C). The packet is next broken into blocks of 12 octets. The block size of 12 octets is used in the unconfirmed data block structure. In this case, the message is transmitted without an acknowledgment of reception from the receiving radio. The fact that the data block structure is unconfirmed also means that the data packet has a 32 bit CRC (cyclic redundancy check)

over the data contents to allow the recipient to determine if the packet has been received without error. This CRC is positioned at the end of the packet in Block m (see Figure II.2). The number of blocks in the packet which follow the header block, the number of pad octets needed to make the last block length 12 octets and the fact that it uses unconfirmed delivery are all indicated in the header block.

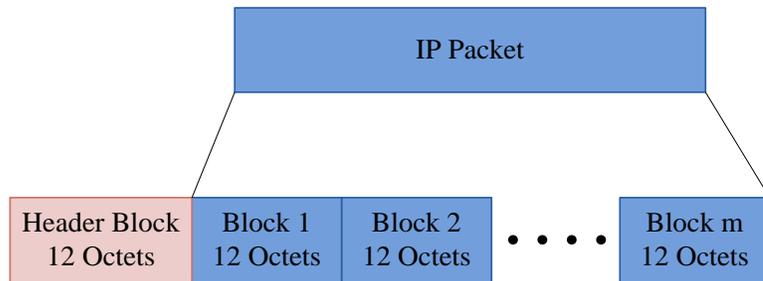


Figure II.2 - Decomposition of Data Message into Short Blocks

**Header Block**

The header block contains 10 octets of address and control information, followed by 2 octets of a header CRC error detection code. The format of an unconfirmed header block is shown in Figure II.3.

Unconfirmed Data Header

|          | Bit 7           | Bit 6            | Bit 5              | Bit 4           | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|----------|-----------------|------------------|--------------------|-----------------|-------|-------|-------|-------|
| Octet 0  | 0               | A/N              | I/O                | 1               | 0     | 1     | 0     | 1     |
| Octet 1  | 1               | 1                | SAP Identifier     |                 |       |       |       |       |
| Octet 2  | MFID            |                  |                    |                 |       |       |       |       |
| Octet 3  | Logical Link ID |                  |                    |                 |       |       |       |       |
| Octet 4  |                 |                  |                    |                 |       |       |       |       |
| Octet 5  |                 |                  |                    |                 |       |       |       |       |
| Octet 6  | 1               | Blocks to follow |                    |                 |       |       |       |       |
| Octet 7  | 0               | 0                | 0                  | Pad Octet Count |       |       |       |       |
| Octet 8  | Reserved        |                  |                    |                 |       |       |       |       |
| Octet 9  | 0               | 0                | Data Header Offset |                 |       |       |       |       |
| Octet 10 | Header CRC      |                  |                    |                 |       |       |       |       |
| Octet 11 |                 |                  |                    |                 |       |       |       |       |

Figure II.3 - Unconfirmed Data Packet Header Format

In the first octet (octet 0), there are two variables located in bits 6 and 5. Since they are both one bit, they can only take on one of two values. A/N defines whether the packet is an unconfirmed data packet (0) or a confirmed data packet (1). Since this application uses unconfirmed data packets, 0 is assigned to bit 6. I/O defines whether the packet is an inbound message (0) or an outbound message (1). Since this application is used to transmit messages from the fixed infrastructure to a mobile client on the common air interface, 1 is assigned to this bit. The SAP Identifier, or service access point identifier, describes the network endpoint used. The SAP Identifier can take on the values shown in Table II.1.

Table II.1 - SAP Identifier Values

|      |                                    |
|------|------------------------------------|
| 0x00 | Unencrypted User Data              |
| 0x01 | Encrypted User Data                |
| 0x02 | Circuit Data                       |
| 0x03 | Circuit Data Control               |
| 0x04 | Packet Data                        |
| 0x05 | Address Resolution Protocol (ARP)  |
| 0x06 | SNDCP Packet Data Control          |
| 0x1F | Extended Address                   |
| 0x20 | Registration and Authorization     |
| 0x21 | Channel Re-assignment              |
| 0x22 | System Configuration               |
| 0x23 | MR Loopback                        |
| 0x24 | MR Statistics                      |
| 0x25 | MR Out-of-Service                  |
| 0x26 | MR Paging                          |
| 0x27 | MR Configuration                   |
| 0x28 | Unencrypted Key Management message |
| 0x29 | Encrypted Key Management message   |
| 0x3D | Non-Protected Trunking Control     |
| 0x3F | Protected Trunking Control         |

The type of message transmitted is a data packet. This means that bits 0 to 5 will take on the hex value 0x04. The next octet contains information pertaining to the manufacturer of the radio receiving the information. The MFID (Manufacturer Identification) has a standard value of 0x00 unless the data packet contains a non-standard channel message. Motorola has a specific MFID (0x90), but through testing it was found that the specific MFID could not be used for unconfirmed data packets. This resulted in the third octet of the header being assigned the value 0x00.

The next three octets are very important to the uniqueness of this thesis. They define the logical link ID. The logical link ID identifies the radio to which the data packet is directed. It is equivalent to the Ethernet Media Access Control (MAC) address on a conventional network. Without the header information from an incoming CAI packet the base station could not respond to the specific client radio requesting information. Since the base station is designed with an analog radio, the whole CAI message will be received, including the CAI header block. The base station will then have information about the specific radio awaiting a response. The demonstration described in this thesis transmits to one specific radio with a predefined logical link id of 054 which corresponds to the hexadecimal value 0x036.

Octet 6 contains information on how many blocks of data follow the header block. This information is used for proper decoding and error checking at the client radio. The Pad octet count in octet 7 describes how many 0x00 octets

are added into the last block to make a complete block of 12 octets. The maximum number of pad octets is 0x08. Octet 8 is a reserved octet which is only used when transmitting and receiving confirmed data packets. Otherwise, the octet is set to 0x00. The data header offset octet is used to divide the data block into a data header and data information. A data header offset is only used in some applications. For this application, the data header offset is set to 0x00.

The last two octets of the header block contain the header CRC. This is the most complex part of the header block formation. It is calculated from the first 10 octets of the header block using the cyclical redundant coding procedure called CRC-CCITT. To calculate the CRC, consider the first 80 header bits (10 octets) of the header block as the coefficients to the polynomial  $M(x)$  of degree 79:

$$M(x) = x^{79} + x^{78} + x^{77} + \dots + x^1 + x^0$$

where  $x^{79}$  is the most significant bit of the zero-th header octet (Octet 0 – Bit 7) and  $x^0$  is the least significant bit of the ninth header octet (Octet 9 – Bit 0). The calculation of the CRC uses two other polynomials. The first is called the generator polynomial and is defined as:

$$G_H(x) = x^{16} + x^{12} + x^5 + 1$$

The second is called the inversion polynomial and is defined as:

$$I_H(x) = x^{15} + x^{14} + x^{13} + \dots + x^2 + x + 1$$

The header CRC polynomial,  $F_H(x)$ , is then computed from the formula,

$$F_H(x) = [x^{16} * M(x) \bmod G_H(x)] + I_H(x)$$

In the above equation, the 'mod' operator stands for the modulus operation computed using modulo 2 arithmetic. Modulo 2 arithmetic uses binary addition with no carries, which is just the exclusive-OR (XOR) operation. After computation,  $F_H(x)$  takes on the form,

$$F_H(x) = x^{15} + x^{14} + x^{13} + \dots + x^1 + x^0$$

where  $x^{15}$  corresponds to the most significant bit of the tenth octet (Octet 10 – Bit 7) and  $x^0$  is the least significant bit of the eleventh octet (Octet 11 – Bit 0).

An example of a completed header block is as follows:

Table II.2 - Example Header Block

|          | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | Hex Value |
|----------|-------|-------|-------|-------|-------|-------|-------|-------|-----------|
| Octet 0  | 0     | 0     | 1     | 1     | 0     | 1     | 0     | 1     | 0x35      |
| Octet 1  | 1     | 1     | 0     | 0     | 0     | 1     | 0     | 0     | 0xC4      |
| Octet 2  | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0x00      |
| Octet 3  | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0x00      |
| Octet 4  | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0x00      |
| Octet 5  | 0     | 0     | 1     | 1     | 0     | 1     | 1     | 0     | 0x36      |
| Octet 6  | 1     | 0     | 0     | 0     | 0     | 0     | 1     | 1     | 0x83      |
| Octet 7  | 0     | 0     | 0     | 0     | 1     | 1     | 1     | 0     | 0x0E      |
| Octet 8  | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0x00      |
| Octet 9  | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0x00      |
| Octet 10 | 0     | 1     | 1     | 1     | 0     | 0     | 0     | 0     | 0x70      |
| Octet 11 | 0     | 0     | 0     | 1     | 1     | 1     | 1     | 0     | 0x1E      |

## **Data Blocks**

The data blocks which follow the header block contain all of the information from the original IP data packet. This includes both the IP header and IP data fields. For the unconfirmed data packet format, there is no CRC or serial number in each of the blocks. Rather, there is one 32 bit CRC in the last data block calculated over all data. The format of the last data block is shown in Figure II.4.

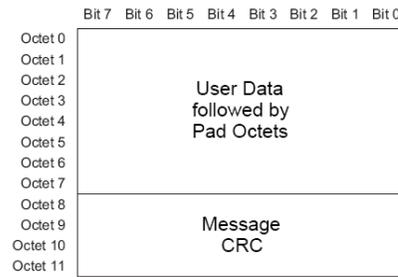


Figure II.4 - Unconfirmed Data Block

The number of Pad Octets is determined by using the following equation:

$$\text{Pad Octet Count} = 12 * \text{Total Number of Data Blocks} - \text{Number of Data Octets} - 4$$

For example, if the data message is of length 17 octets, 2 blocks are needed.

The first data block contains 12 octets of data. The second block is composed of the last 5 octets of data, 3 pad octets and the 4 octets of CRC. However, there is an issue with the Pad Octet Count equation. There are certain scenarios where the pad octet count equals -4, -3, -2, or -1. This occurs when the last data block contains more than 8 octets, but less than 12 octets. The solution to this is not set forth clearly by the standard. Through testing, it was found that when the last block contained between 8 and 12 octets, the remaining octets of the block need to be filled with the value 0x00. The number of data blocks is then increased by one and the pad octet count is automatically set to 8.

The message CRC is a 4 octet cyclic redundancy check coded over all of the data octets after the header block and all of the octets in the last block. The Project 25 standard states that the CRC should be coded over all of the data octets included in the intermediate blocks and the octets of user information of the last block. By this definition, the pad octets should be left out of the CRC

code. However, after testing, it was found that the pad octets of the last block were needed in the calculation of the message CRC. To calculate the CRC, let  $k$  be the total number of user information and pad bits. Consider the polynomial  $M(k)$  to contain all of the bits of user data:

$$M(k) = x^{k-1} + x^{k-2} + x^{k-3} + \dots + x^1 + x^0$$

where  $x^{k-1}$  is the most significant bit of the zero-th octet (Octet 0 – Bit 7) and  $x^0$  is the least significant bit of the last message octet. For the calculation, two polynomials need to be defined. The first is the generator polynomial,

$$G_M(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

The second polynomial needed is the inversion polynomial,

$$I_M(x) = x^{31} + x^{30} + x^{29} + \dots + x^2 + x + 1$$

The message CRC polynomial,  $F_M(x)$ , is then computed using the following formula:

$$F_M(x) = [x^{32} * M(x) \text{ mod } G_M(x)] + I_M(x)$$

The coefficients of  $F_M(x)$  are placed in the last data block with  $x^{31}$  corresponding to the most significant bit of the message CRC (Octet 8 – Bit 7) and  $x^0$  corresponding to the least significant bit of the last message CRC (Octet 11 – Bit 0).

The header block and all the data blocks will now be referred to as the CAI data packet.

## Data Error Correction Code

Now that the complete CAI data packet has been formed, it must pass through data error correction coding. The data blocks for unconfirmed data packets use a rate  $\frac{1}{2}$  trellis code. The encoding process of the rate  $\frac{1}{2}$  code is diagrammed in Figure II.5.

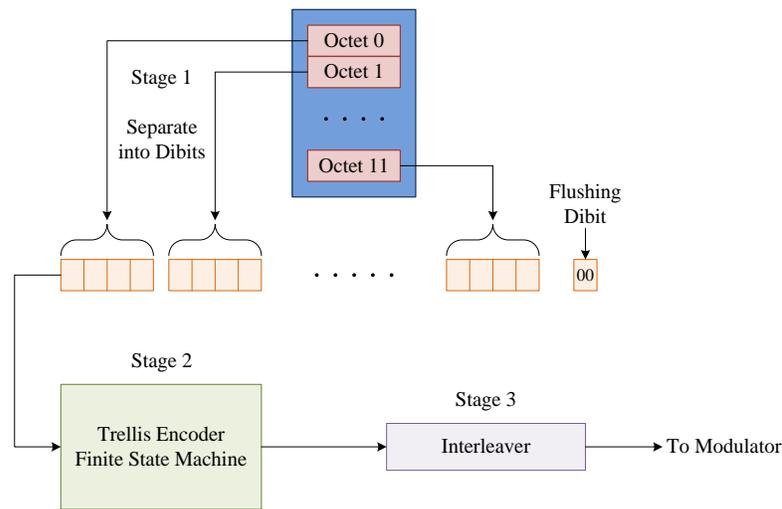


Figure II.5 - Rate 1/2 Trellis Encoder Overview

In the first stage, the octets of the data blocks are separated into dibits. An extra dibit, also known as a flushing dibit (00), is added to the end of the string, to make a total of 49 dibits. This sequence is next passed into stage 2. Stage 2 is a Trellis encoder finite state machine. The trellis encoder receives  $x$  dibits as inputs, and outputs  $2x$  dibits. The encoding process is diagrammed below in Figure II.6.

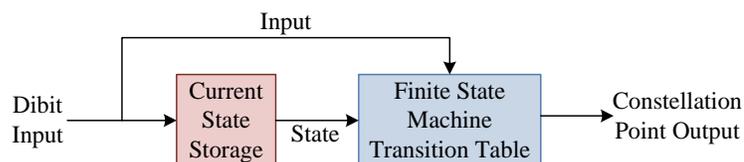


Figure II.6 - Trellis Encoder Block Diagram

The finite state machine uses the current input, previous input, and a state transition table, see Table II.3, to determine a constellation point. For the first input, the previous input, or state, is 00.

Table II.3- Rate 1/2 Trellis Encoder State Transition Table

|              |    | Input |    |    |    |
|--------------|----|-------|----|----|----|
|              |    | 00    | 01 | 10 | 11 |
| FSM<br>State | 00 | 0     | 15 | 12 | 3  |
|              | 01 | 4     | 11 | 8  | 7  |
|              | 10 | 13    | 2  | 1  | 14 |
|              | 11 | 9     | 6  | 5  | 10 |

The constellation point is next mapped to a dibit pair using Table II.4.

Table II.4 - Constellation to Dibit Pair Mapping

| Constellation Point | Dibit 0 | Dibit 1 |
|---------------------|---------|---------|
| 0                   | 00      | 10      |
| 1                   | 10      | 10      |
| 2                   | 01      | 11      |
| 3                   | 11      | 11      |
| 4                   | 11      | 10      |
| 5                   | 01      | 10      |
| 6                   | 10      | 11      |
| 7                   | 00      | 11      |
| 8                   | 11      | 01      |
| 9                   | 01      | 01      |
| 10                  | 10      | 00      |
| 11                  | 00      | 00      |
| 12                  | 00      | 01      |
| 13                  | 10      | 01      |
| 14                  | 01      | 00      |
| 15                  | 11      | 00      |

The input sequence of 49 dibits produces an output sequence of 98 dibits representing a single 12 octet block.

The final stage of data error correction is interleaving. The purpose of the interleaver is to spread burst errors due to Rayleigh fading over the 98 dibit block. In the interleaver, the dibit array is rearranged to form another dibit array according to the interleave table shown in Table II.5.

Table II.5 - Interleave Table

| Interleave Table |             |              |             |              |             |              |             |
|------------------|-------------|--------------|-------------|--------------|-------------|--------------|-------------|
| Output Index     | Input Index | Output Index | Input Index | Output Index | Input Index | Output Index | Input Index |
| <b>0</b>         | 0           | <b>26</b>    | 2           | <b>50</b>    | 4           | <b>74</b>    | 6           |
| <b>1</b>         | 1           | <b>27</b>    | 3           | <b>51</b>    | 5           | <b>75</b>    | 7           |
| <b>2</b>         | 8           | <b>28</b>    | 10          | <b>52</b>    | 12          | <b>76</b>    | 14          |
| <b>3</b>         | 9           | <b>29</b>    | 11          | <b>53</b>    | 13          | <b>77</b>    | 15          |
| <b>4</b>         | 16          | <b>30</b>    | 18          | <b>54</b>    | 20          | <b>78</b>    | 22          |
| <b>5</b>         | 17          | <b>31</b>    | 19          | <b>55</b>    | 21          | <b>79</b>    | 23          |
|                  |             |              |             |              |             |              |             |
| <b>18</b>        | 72          | <b>44</b>    | 74          | <b>68</b>    | 76          | <b>92</b>    | 78          |
| <b>19</b>        | 73          | <b>45</b>    | 75          | <b>69</b>    | 77          | <b>93</b>    | 79          |
| <b>20</b>        | 80          | <b>46</b>    | 82          | <b>70</b>    | 84          | <b>94</b>    | 86          |
| <b>21</b>        | 81          | <b>47</b>    | 83          | <b>71</b>    | 85          | <b>95</b>    | 87          |
| <b>22</b>        | 88          | <b>48</b>    | 90          | <b>72</b>    | 92          | <b>96</b>    | 94          |
| <b>23</b>        | 89          | <b>49</b>    | 91          | <b>73</b>    | 93          | <b>97</b>    | 95          |
| <b>24</b>        | 96          |              |             |              |             |              |             |
| <b>25</b>        | 97          |              |             |              |             |              |             |

To summarize the data error correction portion of the Project 25 standard, a data block of 12 octets is broken into a series of 48 dibits and a flushing dibit is added to the end. This sequence is passed through the 1/2 rate trellis encoder finite state machine which has a constellation point output. The constellation point sequence is mapped to a series of dibit pairs, producing an output sequence of 98 dibits. The last step is to interleave the dibit array. The input to the data error correction is a 12 octet data block and the output is a block of 98 dibits. Once all

the blocks, including the header block, have passed through the data error correction functions, they are placed back together as one long string which will be referred to as the encoded data packet.

**Network Identifier**

The network identifier is placed at the beginning of the encoded data packet. It provides a simple means of addressing radio networks depending on the radio system configuration. The network identifier also has a small field which identifies the type of message, in this case data, so that the proper error correction can be performed. The network identifier encodes 16 bits of information, see Table II.6.

Table II.6 - Network Identifier Information

|                     |     |    |    |    |    |    |    |    |    |    |    |                    |    |    |    |
|---------------------|-----|----|----|----|----|----|----|----|----|----|----|--------------------|----|----|----|
| ↓ Transmitted First |     |    |    |    |    |    |    |    |    |    |    | Transmitted Last ↓ |    |    |    |
| Network Access Code |     |    |    |    |    |    |    |    |    |    |    | Data Unit ID       |    |    |    |
| N11                 | N10 | N9 | N8 | N7 | N6 | N5 | N4 | N3 | N2 | N1 | N0 | D3                 | D2 | D1 | D0 |
| 15                  | 14  | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3                  | 2  | 1  | 0  |

The 16 bits of data are composed of a 12 bit network access code, chosen arbitrarily as 0x054, and a 4 bit data unit ID. The data unit id can take on any one of the values in

Table II.7.

Table II.7 - Data Unit Identifier Values

| Data Unit ID      | Parity Bit | Data Unit Usage                            |
|-------------------|------------|--|
| 0b0000            | 0          | Header Data Unit                           |
| 0b0001 and 0b0010 | 1          | Reserved                                   |
| 0b0011            | 0          | Terminator without subsequent Link Control |
| 0b0100            | 0          | Reserved                                   |
| 0b0101            | 1          | Logical Link Data Unit 1                   |
| 0b0110            | 1          | Reserve                                    |
| 0b0111            | 0          | Trunking Signaling Data Unit               |
| 0b1000 and 0b1001 | 1          | Reserved                                   |
| 0b1010            | 1          | Logical Link Data Unit 2                   |
| 0b1011            | 0          | Reserved                                   |
| 0b1100            | 0          | Packet Data Unit                           |
| 0b1101 and 0b1110 | 1          | Reserved                                   |
| 0b1111            | 0          | Terminator with subsequent Link Control    |

The value for the Data Unit Identifier which will be used for data transmission is 0b1100 with a parity bit of 0. This data unit identifier is used for transmitting a Packet Data Unit.

This information is protected with a (63,16,23) primitive BCH code. A single parity bit is appended to the end to round out the NID code word to 64 bits. The (63,16,23) BCH code is described in the APCO Project 25 standard as being generated using the extension Galois Field. The generator polynomial for the code is of 47<sup>th</sup> degree with 27 non-zero terms. It is given below in octal notation.

$$g(x) = 6631\ 1413\ 6723\ 5453$$

The generator matrix for the full code, after adding the 64<sup>th</sup> bit, is given in Table II.8 in octal notation. The left 16 columns of the code form an identity matrix and the right 48 columns form a parity check matrix.(5)

Table II.8 – BCH Code Generator Matrix

| row | 16 x 16 Identity |      | 16 x 48 Parity Check |      |      |      |
|-----|------------------|------|----------------------|------|------|------|
| 1   | 10               | 0000 | 6331                 | 1413 | 6723 | 5452 |
| 2   | 04               | 0000 | 5265                 | 5216 | 1472 | 3276 |
| 3   | 02               | 0000 | 4603                 | 7114 | 6116 | 4164 |
| 4   | 01               | 0000 | 2301                 | 7446 | 3047 | 2072 |
| 5   | 00               | 4000 | 7271                 | 6230 | 7300 | 0466 |
| 6   | 00               | 2000 | 5605                 | 6507 | 5263 | 5660 |
| 7   | 00               | 1000 | 2702                 | 7243 | 6531 | 6730 |
| 8   | 00               | 0400 | 1341                 | 3521 | 7254 | 7354 |
| 9   | 00               | 0200 | 0560                 | 5650 | 7526 | 3566 |
| 10  | 00               | 0100 | 6141                 | 3337 | 5170 | 4220 |
| 11  | 00               | 0040 | 3060                 | 5557 | 6474 | 2110 |
| 12  | 00               | 0020 | 1430                 | 2667 | 7236 | 1044 |
| 13  | 00               | 0010 | 0614                 | 1333 | 7517 | 0422 |
| 14  | 00               | 0004 | 6037                 | 1146 | 1164 | 1642 |
| 15  | 00               | 0002 | 5326                 | 5070 | 6351 | 5373 |
| 16  | 00               | 0001 | 4662                 | 3027 | 5647 | 3127 |

The network identification code used in this thesis to access the channel used for the transmission and reception of data packets starts as the 16 bit sequence in Table II.9.

Table II.9 - True Network Identifier

| ↓ Transmitted First |    |    |    |    |    |   |   |   |   |   | Transmitted Last ↓ |   |   |   |   |
|---------------------|----|----|----|----|----|---|---|---|---|---|--------------------|---|---|---|---|
| Network Access Code |    |    |    |    |    |   |   |   |   |   | Data Unit ID       |   |   |   |   |
| 0                   | 0  | 0  | 0  | 0  | 1  | 0 | 1 | 0 | 1 | 0 | 0                  | 1 | 1 | 0 | 0 |
| 15                  | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4                  | 3 | 2 | 1 | 0 |

The final 64 bit (8 octet) sequence attached to the beginning of the encoded data packet is:

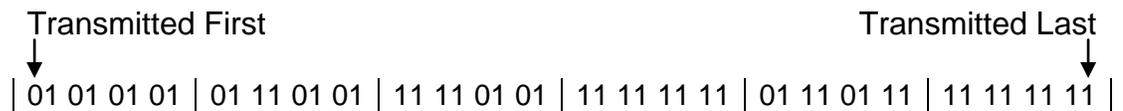
|     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |     |
|-----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|
| MSB | 00 | 00 | 01 | 01 | 01 | 00 | 11 | 00 | 10 | 00 | 10 | 01 | 01 | 10 | 11 | 00  |
|     | 01 | 11 | 01 | 00 | 00 | 11 | 01 | 01 | 11 | 00 | 11 | 11 | 01 | 11 | 11 | 00  |
|     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | LSB |

Note that the network identifier sequence is the same for all transmitted data packets.

## **Frame Synchronization Word**

Frame synchronization is provided by a specific preamble which is known to both the transmitter and the receiver. The frame synchronization word, or sync word, is used to mark the first bit of the message data. It is needed at the beginning of every data transmission. For a transmission over the common air interface, the frame sync word sequence is listed in Table II.10.

Table II.10 - Frame Sync Word Sequence



## **Status Symbols**

Once the frames synchronization (FS) and network identifier (NID) are added to the beginning of the encoded data packet, the message must be expanded with status symbols. The two bit symbols are added after every 70 bits (35 dibit symbols) of the encoded data packet. Status symbol codes are given in Table II.11.

Table II.11 - Status Symbol Codes

| Status Symbol | Meaning                              |
|---------------|--------------------------------------|
| 01            | Inbound Channel is Busy              |
| 00            | Unknown, use for talk-around         |
| 10            | Unknown, use for inbound or outbound |
| 11            | Inbound Channel is Idle              |

Since the full packet data unit, see Figure II.7, is transmitted on a channel which transmits outbound messages and receives inbound messages, the two bit status symbol used is 10.

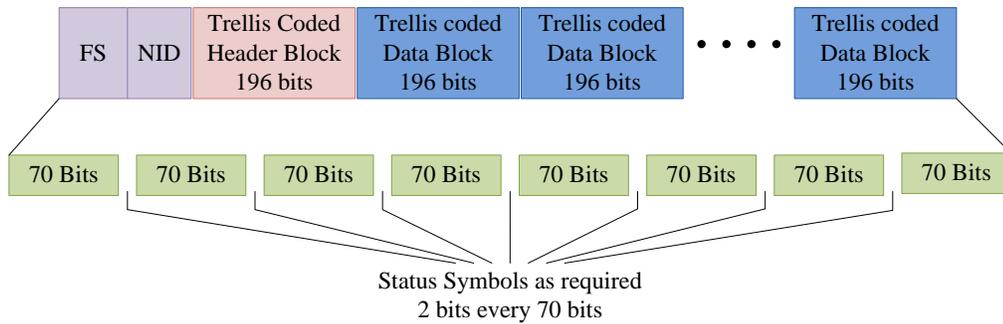


Figure II.7 - Packet Data Unit

### **Channel Initialization**

Through testing, it was found that a transmitted packet data unit would not properly be decoded because of erroneous receive clock synchronization at the beginning of the transmission. To remedy this, the clock synchronization in the receive radio needs to be initialized before the packet data unit, led by the frame synchronization, is transmitted. The clock synchronization signal used is 125 repetitions of the symbol sequence: 01 01 11 11. It was determined by experimenting that this signal is sufficient in tuning the receive radio's decoder so that once the packet data unit is received, it is properly decoded.

### **Dibits to Frequencies**

The channel initialization and packet data unit are both composed of a sequence of dibit symbols. Each dibit symbol is encoded as one of four frequency deviation values. The mapping between bits and frequency deviation values is given in Table I.1.

## Nyquist Raised Cosine Filter

The Nyquist raised cosine filter is also known as a smoothing filter. The role of the smoothing filter is to limit the bandwidth of the frequency deviation signal while minimizing the intersymbol interference. The Project 25 standard specifies this filter should have a frequency response given by:

$$|H(f)| = \begin{cases} 1.0 & f < 1920 \text{ Hz} \\ \frac{1}{2} + \frac{1}{2} * \cos\left(\frac{2\pi f}{1920}\right) & 1920 \text{ Hz} < f < 2880 \text{ Hz} \\ 0.0 & f > 2880 \text{ Hz} \end{cases}$$

A plot of the actual filter gain response, based upon the previous equation, is shown in Figure II.8.

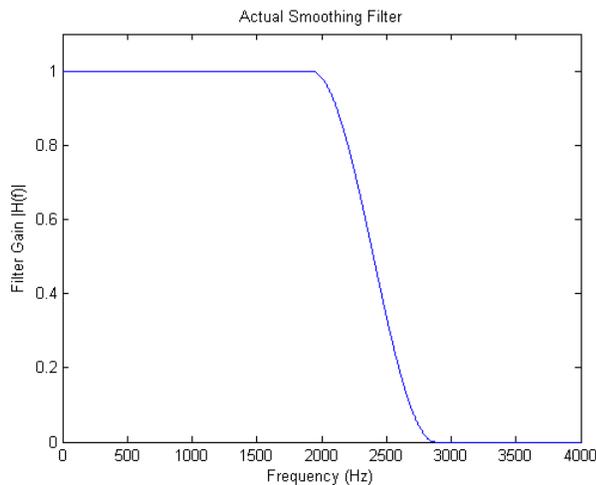


Figure II.8 - Actual Smoothing Filter Frequency Response

The input to the smoothing filter is the sequence of frequency deviation pulses (4800 symbols per second) generated by the 2-bit symbol to 4-level frequency encoder. The normalized impulse response of the smoothing filter is shown in

Figure II.9. The horizontal axis displays time in symbol periods where 1 symbol period is equal to 1/4800 seconds.

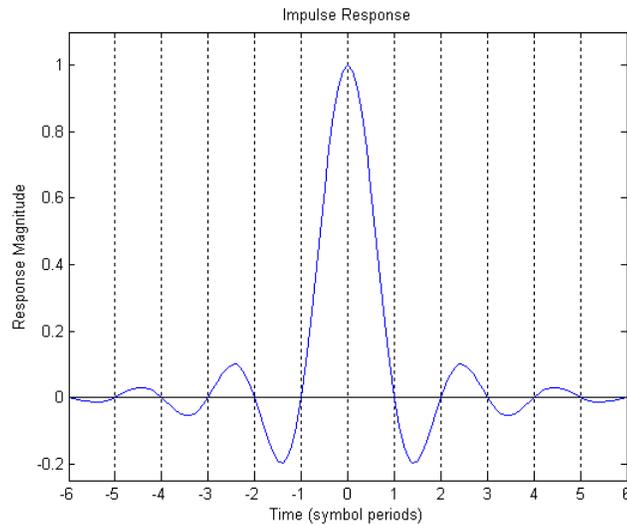


Figure II.9 - Impulse Response of the Smoothing Filter

When this filter impulse is convolved with the input sequence, the effect of each symbol pulse is spread by the filter over approximately 12 symbol periods, 6 before and 6 after the main peak. This means that the filter output reflects not just a single pulse filter, but is affected by the previous several impulses. However, it can be observed that the zero crossings of the impulse response happen at integer multiples of the symbol periods. By properly spacing the frequency deviations, the frequency deviation needed can occur at relative time 0, while the 6 symbol periods before and after fall at the zero crossings and are nullified. The smoothing filter is implemented digitally at an effective rate of 48000 samples per second. To do this, the input frequency deviations are padded by 9 zeros so that they occur every tenth symbol.

The designed smoothing filter is a truncated version of the actual or ideal smoothing filter. The time base for the impulse response of the ideal filter, Figure II.9, reaches from negative infinity to positive infinity. Due to computational restraints, the impulse response is truncated at -6 and +6 symbol periods. This results in a filter with 120 coefficients instead of an infinite number of coefficients. The designed smoothing filter response versus the actual smoothing filter response is shown in Figure II.10.

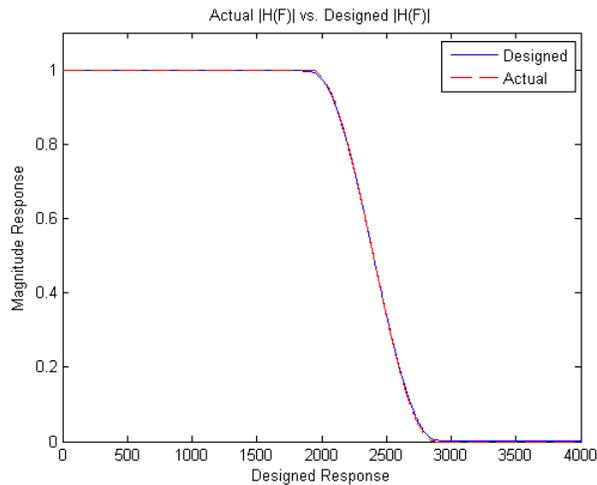


Figure II.10 - Actual and Designed Magnitude Response

An example of the input and the output of the smoothing filter is shown below in Figure II.11.

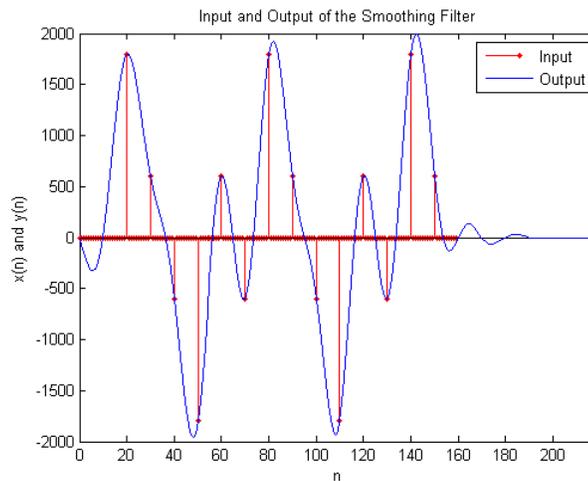


Figure II.11 - Example of Input and Output of the Smoothing Filter

The red stem plot shows the input frequency deviations. Note that the frequency deviations are padded by 9 zeros.

### **Pre-Emphasis Filter**

The output of the smoothing filter is now passed on as the input to the pre-emphasis, or shaping, filter. The role of this filter is to pre-compensate for filtering which is performed in the symbol decoder section of the client digital radio. To understand the shaping filter, the built-in decoding filter of the client digital radio must be explained. The decoding filter is a ten term (one symbol period) running average filter. This running average filter has the frequency response shown in Figure II.12.

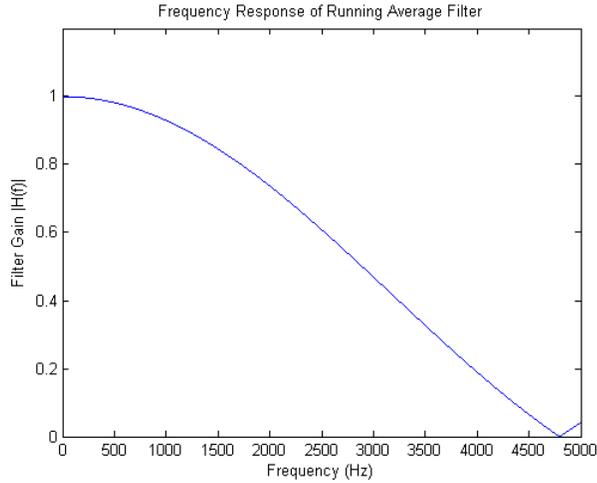


Figure II.12 - Running Average Filter Frequency Response

The equation for the filter gain is as follows:

$$|H(f)| = \frac{\sin\left(\frac{\pi f}{4800}\right)}{\frac{\pi f}{4800}} = \text{sinc}\left(\frac{\pi f}{4800}\right)$$

The shaping filter is intended to pre-compensate in the transmitter for this attenuation in the receiver. Thus, the appropriate filter gain should be the inverse of the equation for the decoding filter:

$$|H(f)| = \frac{\frac{\pi f}{4800}}{\sin\left(\frac{\pi f}{4800}\right)} = \frac{1}{\text{sinc}\left(\frac{\pi f}{4800}\right)}$$

The ideal gain of the shaping filter goes to infinity whenever the frequency is an integer multiple of 4800 Hz. However, the shaping filter is preceded by the smoothing filter. Since the smoothing filter has zero gain for all frequencies above 2880 Hz, the shaping filter will always be in the range of stability. Figure II.13 shows the ideal shaping filter for the inverse gain function.

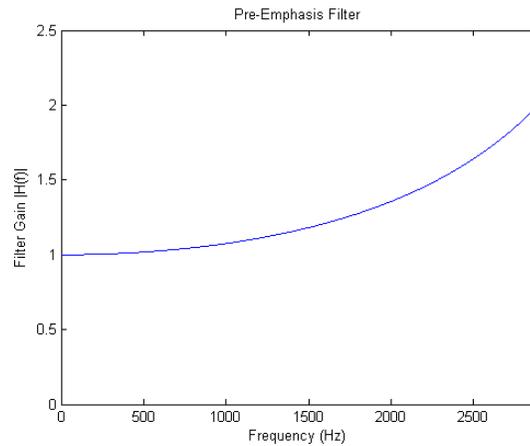


Figure II.13 - Ideal Pre-Emphasis Filter

To design this filter, the exact inverse function for  $|H(f)|$  could not be used. Instead, the filter was designed as a low pass filter based on the characteristics of an inverse sinc filter. The tool used to aid in the design of the filter coefficients was MATLAB's Filter Design Toolbox. The resulting filter is an FIR (finite impulse response) filter with 39 filter coefficients. See Appendix B. The filter gain of the designed filter has the same characteristics as the ideal filter for  $0 \leq f \leq 2880$ . However, for  $f > 2880$ , the filter takes on the characteristics of a low pass filter. This is done so that the designed filter is always stable. The designed filter gain versus the actual filter gain is shown in Figure II.14.

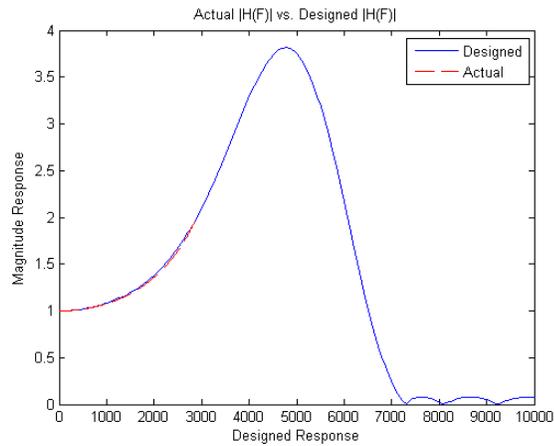


Figure II.14 - Magnitude Response of the Actual and Designed Pre-Emphasis Filter

The output of the pre-emphasis filter is a fully encoded and modulated Project 25 Data Packet in the form of an analog signal sampled at 48000 samples per second.

### **Push-to-Talk Control**

Before the Project 25 waveform can be transmitted through the analog radio, the push-to-talk (PTT) of the radio must be pressed. The PTT is an internally powered, normally high, 5V, signal. This is an active low signal, meaning that the PTT is triggered when the signal is grounded. To ground this signal, one of the RS-232 standard control lines was used. The control line used was the Data Terminal Ready (DTR) line. This is accessed through pin 4 of a standard RS-232 DB-9 connection. In code, the communication (COM) port is enabled and handshaking is disabled. The DTR line is normally low, which means that after initialization, the control for the DTR needed to be set true. Before a transmission of audio, the DTR control is set to false, or 0, which sets

the voltage of the line to -9 V. This activates the PTT of the radio. Once the transmission ends, the DTR value returns to true, which brings the voltage up to +9 V and disables the radio PTT. Since the radio PTT input is designed for an open circuit or ground, rather than an active drive of +9V or -9V, the input was protected using a simple passive circuit (Appendix F).

### **Sound Card Control**

The sound card is controlled through a standard Windows library called winmm.lib. This library allows for the setup and control of basic playback features through the computer's default soundcard. For this application, the sound card is opened with a sampling rate of 48 KHz, the sound is played at 100% volume, and a function waits for the sound to be played.

## CHAPTER III

### TESTING

#### Testing Configuration

Testing was a crucial part to the completion of the Project 25 CAI base station. It occurred during key points of the software and hardware development. The equipment test configuration is shown in Figure III.1.

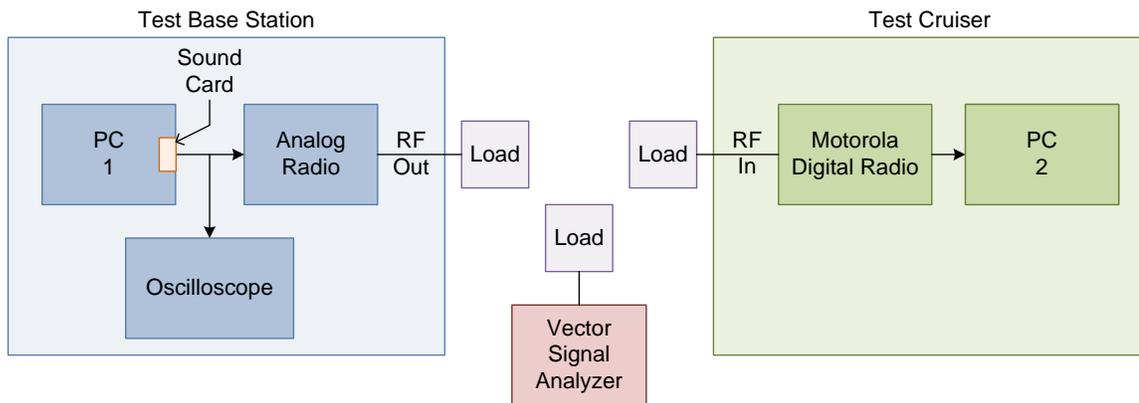


Figure III.1 - Testing Configuration

The test configuration can be segmented into three main sections. These sections are only connected through radio frequency (RF) transmissions. The first section is the test base station. The test base station is comprised of a PC, with a sound card, and an analog radio. The sound card used is a Creative Sound Blaster X-Fi card. This specific sound card was used because of its ability to play audio at a the desired rate of 48 KHz, its high signal to noise ratio, and its linear frequency response. The output of the sound card is attached to the

input of an analog radio. The radio frequency output of the analog radio is attached to a load.

Another section of the test configuration is the Vector Signal Analyzer (VSA). The VSA used was an Agilent 89441A with an Intermediate Frequency (IF) unit and a Radio Frequency (RF) unit. For this setup, a specific Agilent module (Option AYA) was installed. This option added vector modulation analysis for characterizing digital modulation schemes, in particular APCO 25. The VSA was tied to the rest of the setup through a load attached to the RF Section.

The final section of the test configuration is the test cruiser section. The test cruiser section mimics the setup found in a New Hampshire local police cruiser. It has a Motorola XTL5000 digital radio attached to a PC running Windows XP. The digital radio communicates with the PC through the Project54's Project25PPP service.

### **Software Testing**

The first set of tests was conducted after the TxRx application was complete. These tests were all conducted at the software level. The first step was to print all of the header block and data block information to a command window. The results of this test allowed for the correction of code issues such as the pad octet count calculation, calculation of the header block CRC and the data blocks CRC, and the detection of issues with the Data Error Correction function.

The next set of tests was conducted on the designed filters. This was done by printing the input and output of each filter into a text file. The text files were then parsed and plotted using MATLAB. The input and output of the smoothing filter is shown in Figure III.2.

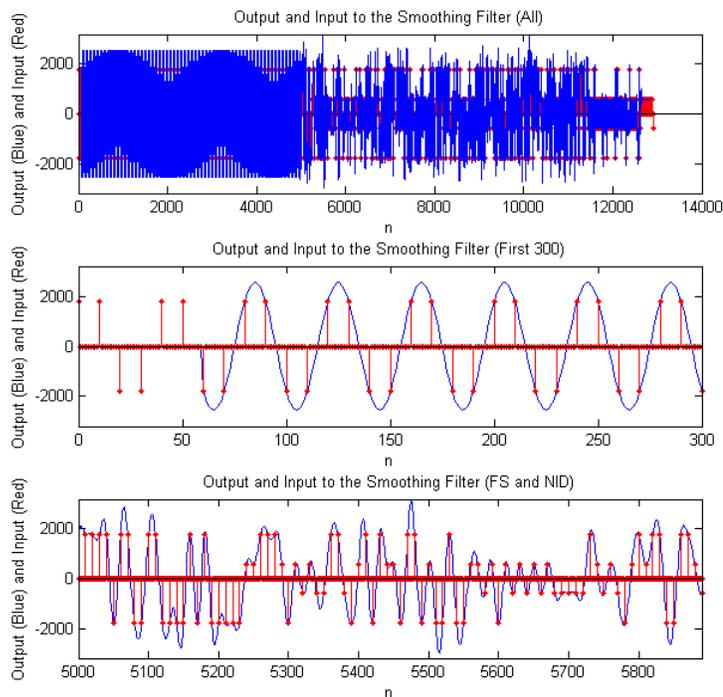


Figure III.2 - Output and Input to the Smoothing Filter (Vertical Axis in Hz)

The red stem plot denotes the frequency deviation inputs to the smoothing filter. The top plot shows the whole array of inputs and outputs. The middle plot shows the first 300 points of the array of inputs and outputs. Note that the input is the alternating symbols discussed in the Channel Initialization section of Chapter II. The last plot shows the frame synchronization and network identifier discussed in the Frame Synchronization and Network Identifier sections of Chapter II. By

seeing the desired signal output of the smoothing filter, it can be assumed that the calculation for the convolution of the frequency deviations with the filter coefficients is correct.

The next series of plots, see Figure III.3, shows the output of the smoothing filter as the input to the shaping, or pre-emphasis filter. The same coloring scheme is used, where the input is shown in red and the output is shown in blue.

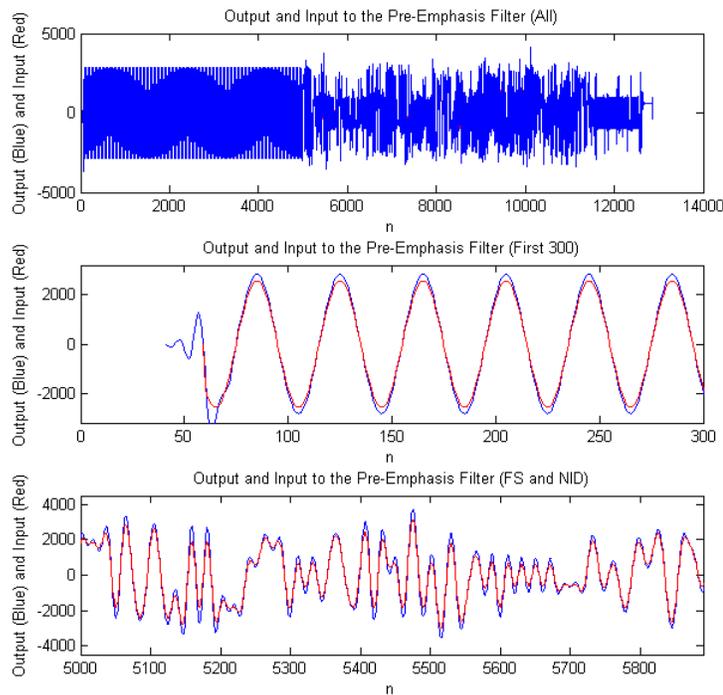


Figure III.3 - Output and Input to the Pre-Emphasis Filter (Vertical Axis in Hz)

The output of the filter puts more emphasis on the frequencies closer to positive and negative 2880 Hz. The above plots are used for testing in the same way that the smoothing filter plots were used. The desired relationship meant that the

calculation of the convolution of the output of the smoothing filter with the pre-emphasis filter coefficients is correct.

### **Sound Card Testing**

The next step in the testing process involved viewing the output of a test signal on an oscilloscope.

The Project 25 CAI Standard uses a test sequence with a known modulation output. The test sequence is the same as the channel initialization sequence. The output of this test sequence should be a sine wave with a frequency of 1.2KHz. The output of the sound card, with frequency deviations equal to the channel initialization sequence, is shown in Figure III.4.

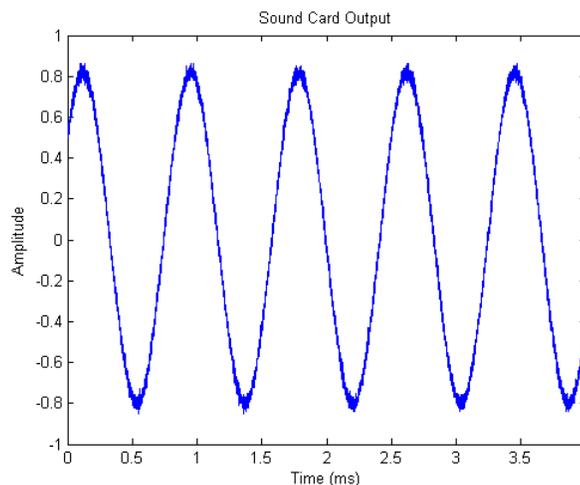


Figure III.4 - Oscilloscope Reading of Test Signal

Note that this signal has the same shape as the signal in the center plot of Figure III.2.

Once the sine wave in Figure III.4 was correctly viewed, a more complicated signal was captured. The signal is the frame synchronization. The

ideal pre-compensated synchronization frame is the first 24 symbols of the last plot in Figure III.2. The output of the sound card is shown below in Figure III.5. Note that the shapes of the two waves are identical. The only difference is in the units used to measure the amplitudes.

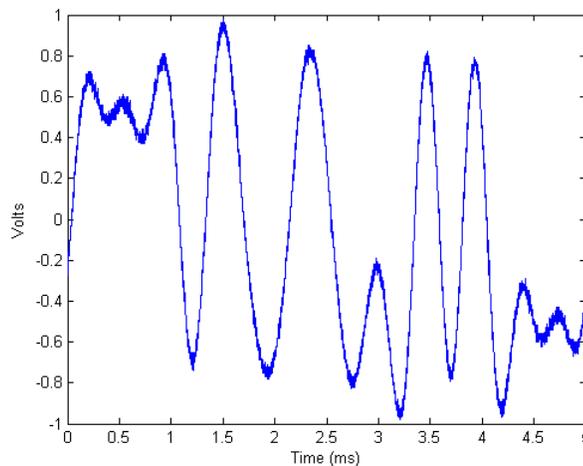


Figure III.5- Oscilloscope reading of the Frame Sync.

### **Analog Radio Modulation Testing Using A VSA**

For the next step in the testing process, a vector signal analyzer (VSA) was used. A vector signal analyzer is a version of a spectral analyzer which can also be used for some digital demodulation applications. The specific vector signal analyzer used for testing, an Agilent 89441A, had the capability of receiving a radio frequency (RF) transmission and demodulating it. The VSA also had a module, Option AYA, which could be used to demodulate digital Project 25 CAI Packet transmissions. The first step in testing with the VSA was to adjust the amplitude of the signal used as the input to the analog radio. The amplitude was adjusted using the test sequence shown in Figure III.4. The

modulated signal, received by the VSA, should be a 1.2 KHz signal with a peak frequency deviation of 2880 Hz. The frequency of the signal was verified using the VSA to produce a frequency spectrum which was centered around the carrier frequency of 155.37 MHz. See Appendix D for the VSA settings to produce the result shown in Figure III.6.

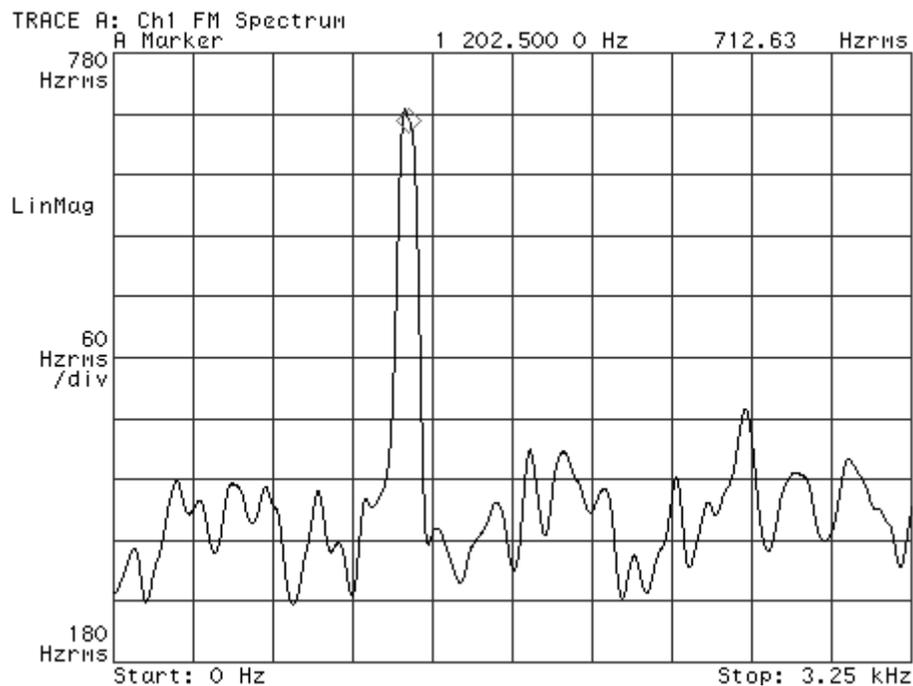


Figure III.6 - Frequency Spectrum using VSA

Note that there is a prominent spectral peak at approximately 1.2 KHz. For the measurement of the peak frequency deviation of the received signal, a plot of frequency versus time was viewed on the VSA. It was found that the amplitude of the signal being passed to the sound card, in the test base station section, was not great enough to produce the desired peak frequency deviation. By increasing the amplitude of the signal by a factor of 3, the peak frequency deviation fell into the range of 2880 KHz.

The vector signal analyzer was next used for testing the analog radio. A new test signal was defined. The signal which was used for testing was a repeated series of eight 1800 Hz symbols and eight -1800 Hz symbols. The test sequence is shown in Figure III.7.

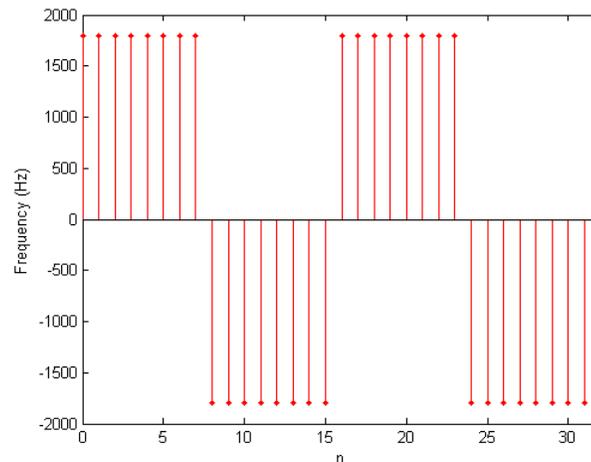


Figure III.7 - New Test Sequence

The first radio used to transmit the new test signal was a Motorola Astro, the same style as the radio in the test cruiser section. The Motorola digital radio had the capability of programming a channel in either analog or digital mode. The external input used to connect the computer audio output to the radio's audio input was the auxiliary microphone (AUX\_TX) and auxiliary push-to-talk

(AUX\_PTT)

pins.

See

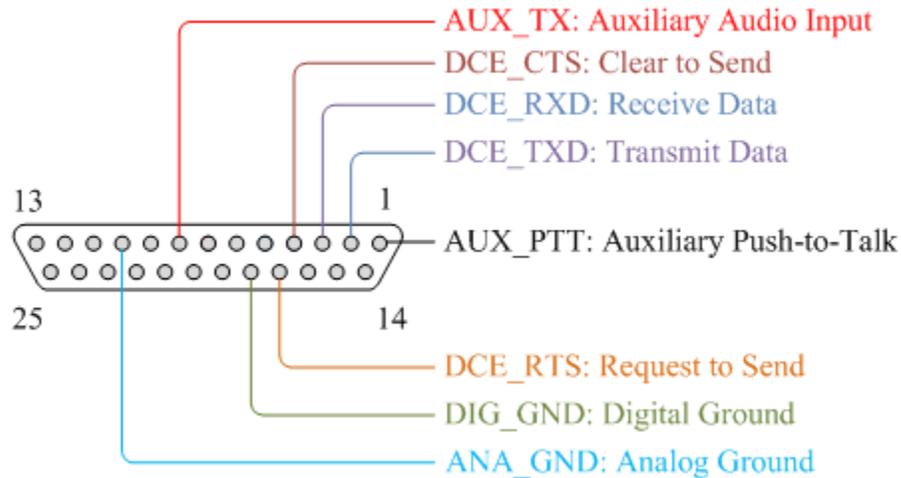


Figure III.8.

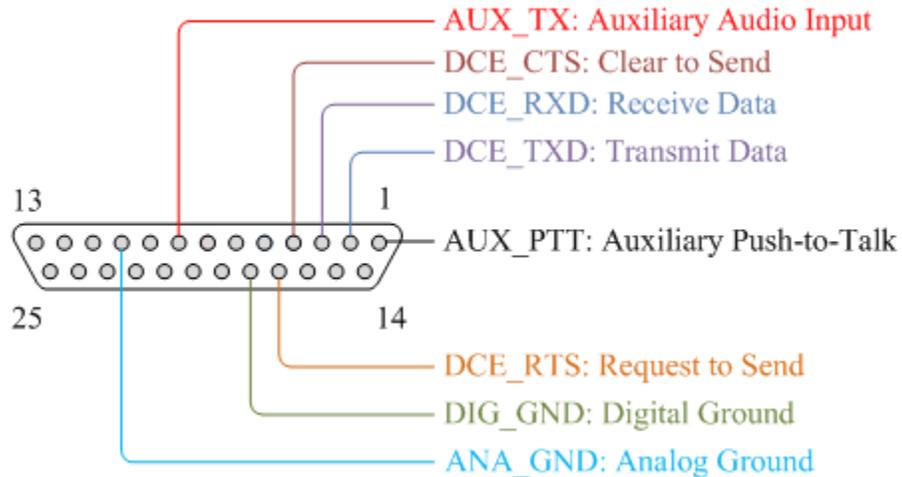


Figure III.8 – Female DB-25 Connector on Motorola Astro

However, it was found that the AUX\_TX pin was not a direct input to the frequency modulator. Instead, the input signal went through sampling and filtering. When the new test signal was transmitted, the VSA showed that the transmitted signal was actually a distorted version of the true signal. See Figure III.9.

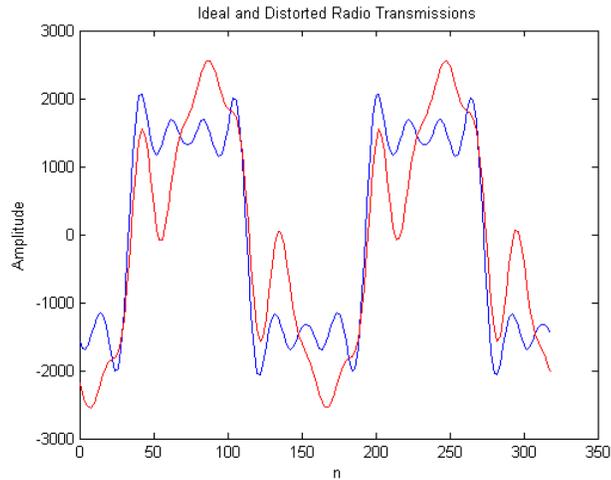


Figure III.9 - Ideal and Distorted Radio Transmissions

The ideal transmitted signal is shown in blue, while the distorted version is shown in red. There is a correlation between the two signals, but the error is too large. After viewing the schematics of the Motorola ASTRO, it was found that there was no way to avoid the sampling and filtering.

The Kenwood TK-7180 analog radio was found to have a direct line to the frequency modulator with a specific push-to-talk signal used for the direct line. The input and output connections to the Kenwood TK-7180 occurs through a 25 pin connector on the back of the radio. The pin out of the connector is shown in Figure III.10.

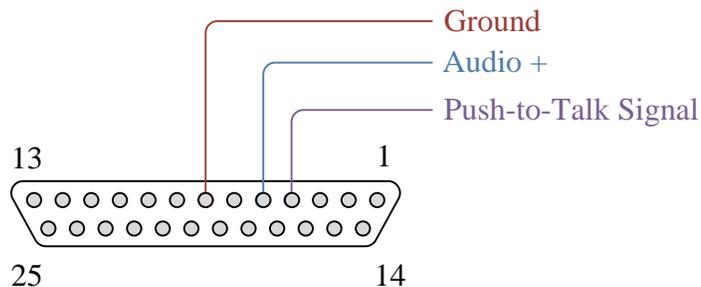


Figure III.10 - Female DB-25 Connector on Kenwood TK-7180

When the test signal was transmitted through the Kenwood radio, the resulting waveform was found to be very similar to the ideal test signal. Due to this result, the Kenwood TK-7180 would be used through the next series of tests.

The next step in the testing of the analog radio was to use the Project 25 digital demodulation module (Option AYA) to test the transmission of a true Common Air Interface Packet. The VSA digital demodulation module makes the VSA act like the Quantar Station discussed in Chapter I. The input to the VSA is the FM transmission and the displayed output is a dibit stream. Figure III.11 shows the demodulated frame synchronization and Figure III.12 shows the corresponding dibit sequence. The settings to achieve the following figures are given in Appendix D.

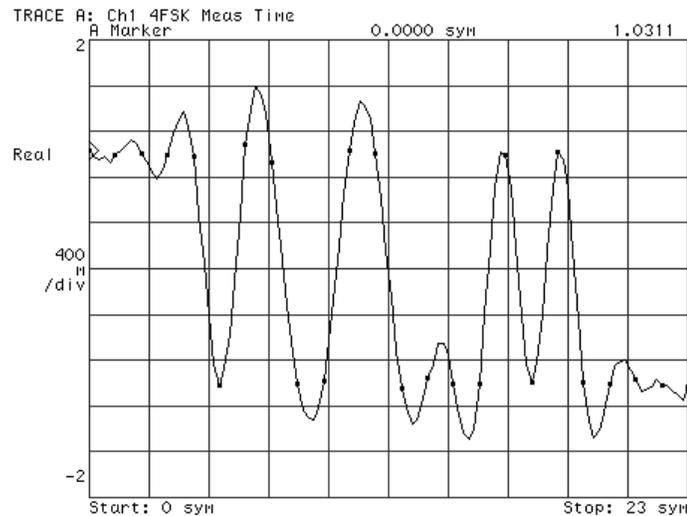


Figure III.11 - APCO 25 Demodulation (Time Measure) using VSA

Note that the demodulation of the transmitted signal is similar to the ideal frame synchronization waveform in the lower plot of Figure III.2.

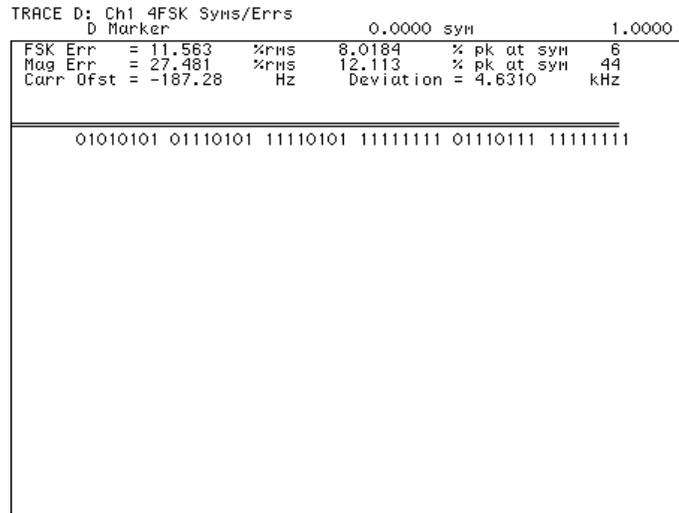


Figure III.12 - APCO 25 Demodulation (Symbols/Errors) using VSA

The dibit stream representing the demodulated signal, in Figure III.12, is the same dibit stream as the ideal frame synchronization word found in Table II.10.

### **Motorola Digital Radio Demodulation**

Once the signal was properly being demodulated by the vector signal analyzer, it was time to test if the Motorola Astro radio would demodulate and decode the transmitted message. Refer to Appendix A for Motorola code plug settings. At first, when the message was transmitted through the analog radio, the “Busy” light on the Astro would activate, but no message was present at the serial data output. This meant that the radio saw activity on the frequency channel, but it was ignored. This was because the logical link ID was incorrect. Once this was fixed, the transmitted message caused the radio to display the message “DATA TRX”. However, no information was received at the output. It was found that the CRC was calculated incorrectly. The issue was found and fixed. After this fix, messages were transmitted and received correctly.

## **End-to-End IP Testing**

Since messages could be transmitted and received correctly, the robustness of the setup needed to be tested. To do this, the P25Messenger was used.

The P25Messenger application was written as a demonstration and testing tool to show the capabilities of the TxRx application. This application is a simple IP messenger. The message is typed on the transmit end and transferred to the receive end via IP protocol. Since it uses standard IP messaging techniques, it can also be used, in conjunction with the IP Reflector and TxRx application, to transmit messages across the common air interface.

Once the message is typed and the “Return” key is pressed, the message is encapsulated in an IP packet and sent to the address 10.1.0.2. This message runs through the IP reflector and is passed to the TxRx application. Note that according to Figure C.1 in Appendix C, the P25Messenger is an “IP Network Application” and the TxRx application is “Custom Data Modem Software”. The message is sent through the CAI and is demodulated and decoded by the Motorola Astro radio. The output of the Astro radio is an IP packet bound for address 10.1.0.2. This was the original payload prior to running through the TxRx Application. The receive side of the application is listening at a specific port and IP address, which is the same port and address to which the transmit side is sending messages. The IP packet is received and displayed in the P25Messenger application on the receive side. Messages of various lengths

were transmitted and received across the common air interface. Screen shots of the P25Messenger are shown below in Figure III.13 and Figure III.14.

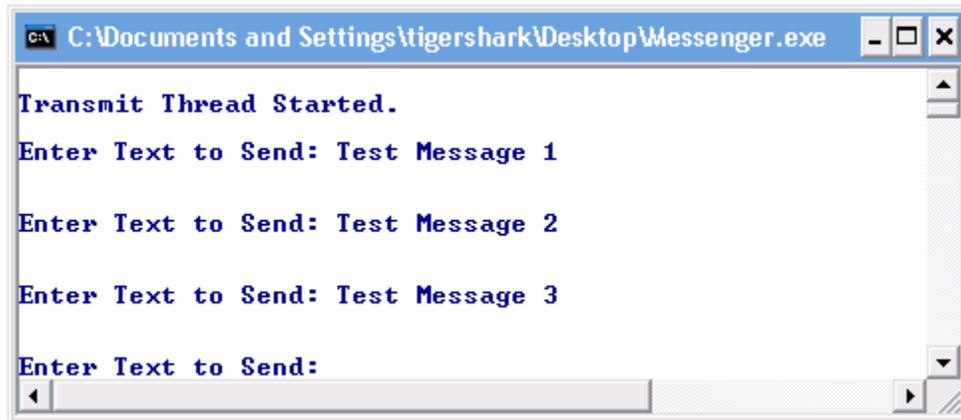


Figure III.13 - Transmit Side of P25Messenger

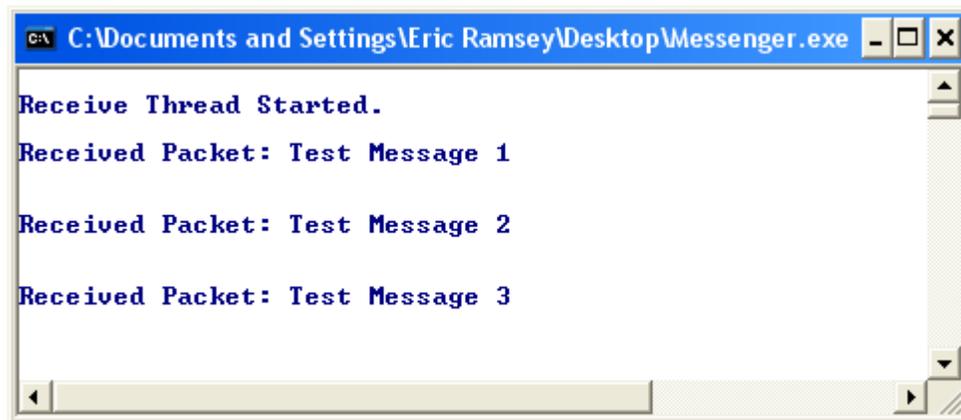


Figure III.14 - Receive Side of P25Messenger

Now that the messaging system worked correctly, a more thorough testing of the setup needed to be conducted. the code for the P25Messenger was rewritten to transmit a count message. The structure of the transmitted message was: "Transmit Message Count: x" where the variable x took on the value of the transmission counter. The receive end of the P25Messenger also had a counter

for the number of messages received. By comparing the number of messages received with the number transmitted, it was possible to determine the number of transmitted packets lost. The P25Messenger transmitted and received on two separate computers. The result of this test was a total transmit count of 13,066 messages (transmitted at 5 second intervals) and total receive count of 13,053 messages. Thirteen messages (0.0099% of transmitted messages) were not successfully received. In trying to determine the cause of the errors, it was found that the frequency of occurrence of errors increased when the receive computer was made busy by running unrelated, CPU intensive, software. This leads to the hypothesis that Project54's Project25PPP service might occasionally have not been getting the CPU time it needed to properly handshake with the Motorola radio. Since this software was not developed specifically for this thesis and the overall packet loss was very low, this will be investigated more fully in the future.

## CHAPTER IV

### CONCLUSIONS

The primary goal of this thesis was to design and develop a cost effective way for local departments to transmit information from headquarters to cruisers. The result was the Project 25 Analog Base Station. Using the pre-existing radio network infrastructure, which is standardized across all local departments in the state of New Hampshire, IP packets were transmitted in the form of CAI data packets. The source of the transmissions was a desktop computer attached to an analog radio. These two components make up the Project 25 Analog Base Station as shown in Figure IV.1.

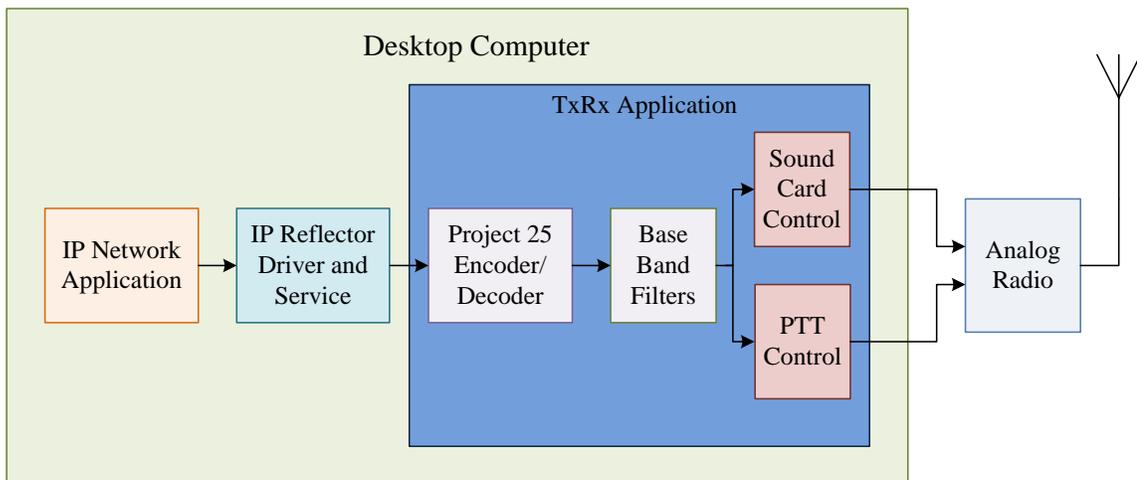


Figure IV.1 - Overview of the Project 25 Analog Base Station

The desktop computer runs a custom application, designed and written for this thesis, called the TxRx Application. The TxRx Application receives IP Packets from the IP Reflector Service, encodes the packet as the data block of a

CAI data packet, filters with a Nyquist Raised Cosine filter and a Pre-Emphasis filter, and then transmits the waveform through the computer's sound card. The output of the sound card is then fed through a custom cable and used as the audio input of an analog radio. The analog radio transmits the audio on a pre-determined CAI digital channel. A digital radio is tuned to this frequency and receives the CAI Data packet. The digital radio demodulates and decodes the data packet which results in the IP Packet which was first sent to the TxRx Application.

The system developed in this project implements the transmission half of a Project 25 base station data radio (Figure I.6). A separate effort to develop the receive portion is ongoing. Once the base station is able to receive requests for information, it can be used for record queries, through Project54, by local police. This would result in a safer, more time efficient method of requesting vehicle and driver information. Since all local police agencies in New Hampshire already have Project 25 data capable mobile radios in their cruisers, this approach would be cost effective as well.

Since the receive portion of the base station data radio was not yet available, the system developed could not be tested in a normal operating configuration. However, the data transmission system was tested in each stage of development against the standards set forth in Project 25 documents. Subsequently, the transmitter was tested in terms of end-to-end IP Packet delivery using a commercial Project 25 compliant digital receiver (Motorola Astro

Digital Radio). The fact that all of these tests yielded positive results, gives high confidence that the design of the transmission system is sound.

### **Future Improvements**

There are improvements that can be made to the design and implementation of the Project 25 analog base station.

One of the improvements is to only transmit when the channel is not busy. Since all of the tests were conducted on one radio transmitting on a specific frequency, there was no need to wait for the channel to be available. However, if the desire is to perform mixed voice and data on the same channel, there is a possibility of lost CAI Data packets. The loss of packets would be caused by transmission collisions between the data and voice base station radios. This improvement would be used to increase the robustness of the system.

Another improvement is to implement dynamic mapping of mobile radio logical link IDs to IP addresses, in cooperation with the receive side of the Project 25 analog base station. For the current setup, the logical link ID is coded into the TxRx application. So in effect, all messages are addressed to the same mobile radio. Once a request for information message is received, the logical link ID of the mobile radio should be stored in a table of values, matched to its IP address. When the return information is received over the IP network, it can be sent to the correct radio.

Once the receive side is complete, the TxRx application should be edited so that it can run as a Windows service instead of an application. This way, a

console application would not have to be running in the background on the server.

The TxRx application should also be changed to implement the confirmed mode of packet communication. This would involve a difference in the structure of header and data block formations, including block length, as well as different CRC computations. Confirmed data packet transmissions would also need to be able to provide handshaking with the receive radio to make sure the data packet was properly received.

Another interesting implementation, which involves the reception of digital data packets, is CAI data registration. With CAI data registration, a digital radio sends a message to the base station when it is on a data capable channel. The message contains identification information including the IP address and logical link ID of the computer. The logical link ID corresponding to the radio could then be accepted and added to the dynamic table of values, or rejected, disabling the radios capability to transmit data onto the channel.

## REFERENCES

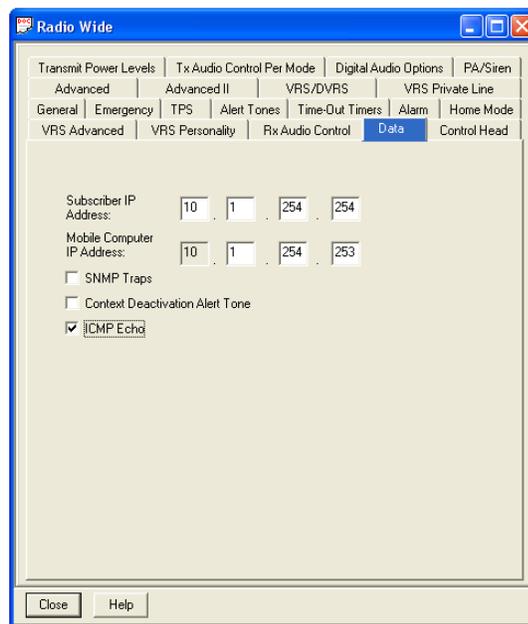
1. **CATLab - UNH.** Consolidated Advanced Technologies for Law Enforcement Program (CATLab). [Online] [Cited: March 11, 2007.] [www.project54.unh.edu](http://www.project54.unh.edu).
2. **Etherstack, Inc.** *APCO P25 Base Station*. Amsterdam : Etherstack, Inc.
3. **Chiu, Andrew and Forbess, Jessica.** *A Handheld Software Radio Based on the IPAQ PDA: Software*. Orlando, FL : Software Defined Radio Forum Technical Conference, 2003.
4. **Digital Voice Systems, Inc.** AMBE-2000 Vocoder Chip. [Online] [Cited: May 7, 2007.] [www.dvsinc.com](http://www.dvsinc.com).
5. **Telecommunications Industry Association.** *Project 25 FMA Common Air Interface New Technology Standards Project Digital Radio Technical Standards*. Arlington : Telecommunications Industry Association, 1998. TIA/EIA-102.BAAA.
6. **Motorola, Inc.** *ASTRO Digital XTL5000 Basic Service Manual*. Fort Lauderdale, Florida : s.n., 2005.
7. **Kenwood, Inc.** *TK-7180 Service Manual*. Japan : s.n., 2004. B51-8691-00.
8. **Stallings, William.** *Data and Computer Communications - Seventh Edition*. Upper Saddle River : Pearson Education, Inc., 2004. 0-13-100681-9.
9. **Comer, Douglas E.** *Internetworking with TCP/IP Vol. 1. - 4th Edition*. s.l. : Prentice Hall, 2000. 0-13-018380-6.

## APPENDIX A

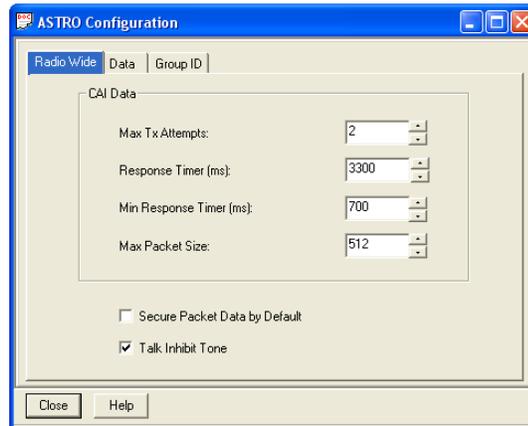
### MOTOROLA XTL5000 CODE PLUG

The Motorola XTL5000 code plug settings needed for CAI Data packet reception are edited using the programming software provided by Motorola.

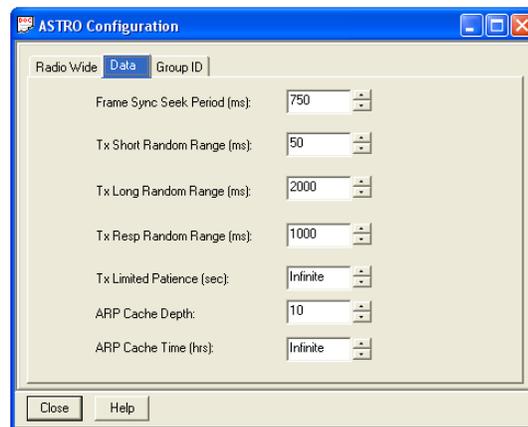
Under Radio Configuration – Radio Wide – Data:



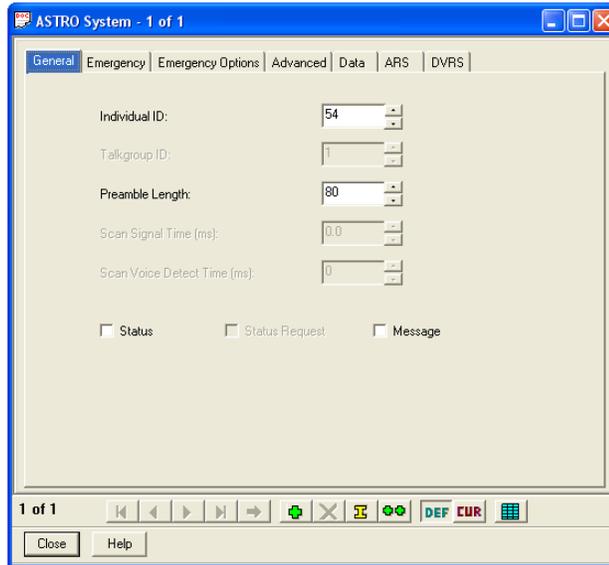
Under Conventional – Astro Systems – Astro Configuration – Radio Wide:



Under Conventional – Astro Systems – Astro Configuration – Data, keep all the factory defaults:

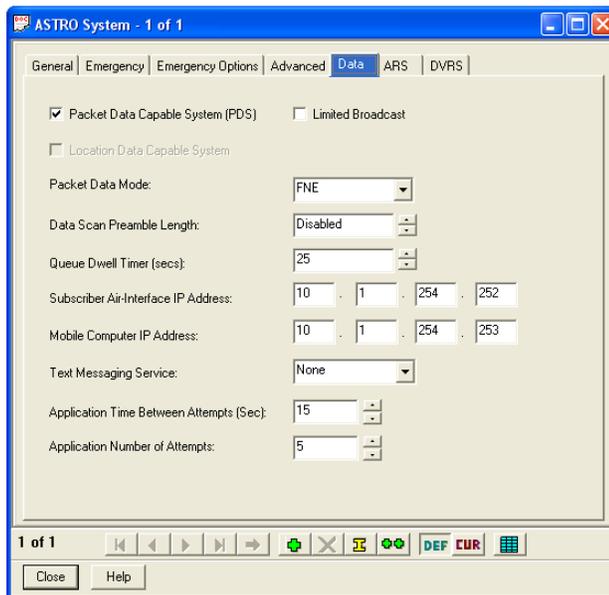


Under Conventional – Astro Systems – Astro System – Astro System 1 – General:

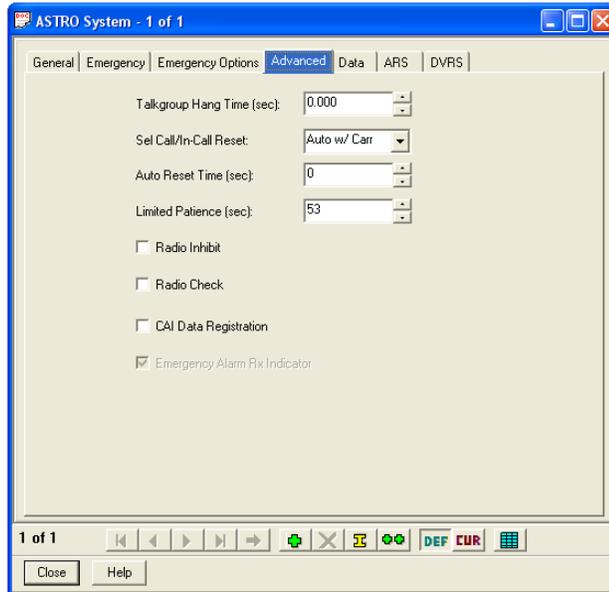


Note that the Individual ID is set to 54. This is the same as the logical link ID set in the header of the transmitted data packet.

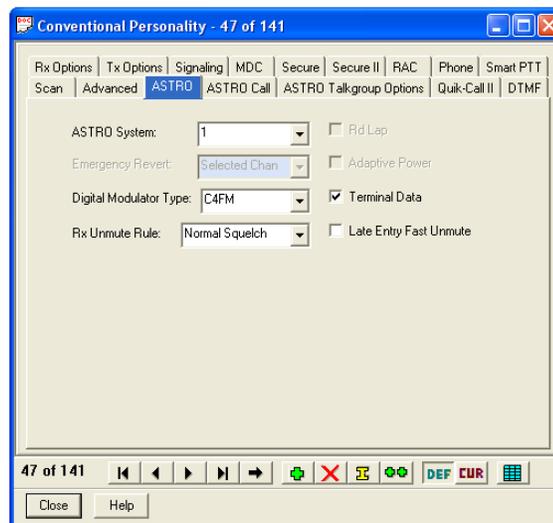
Under Conventional – Astro Systems – Astro System – Astro System 1 – Data:



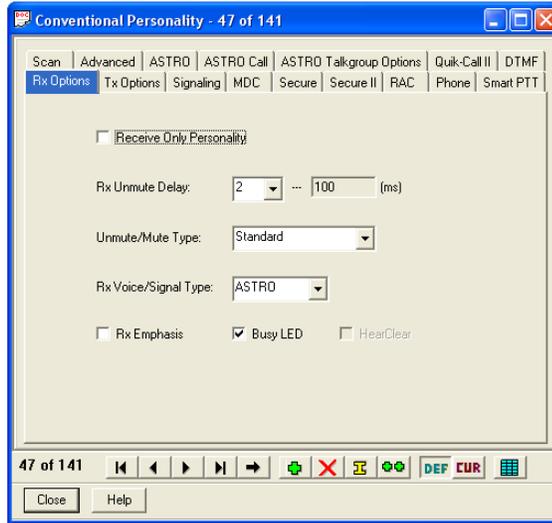
Under Conventional – Astro Systems – Astro System – Astro System 1 – Advanced:



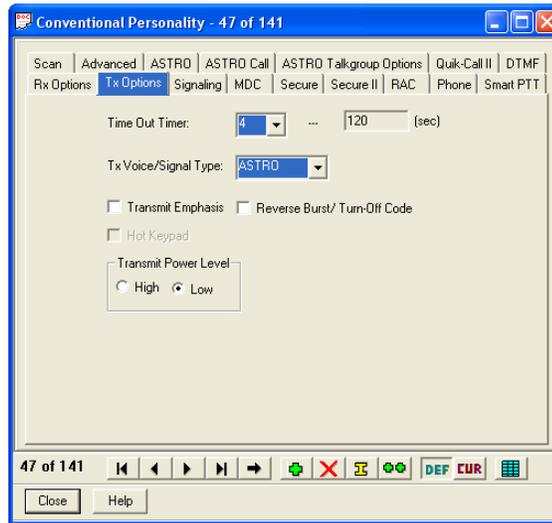
Under Conventional – Conventional Personality – Conventional Personality n ( the configured channel) – Astro:



Note that for the test setup, the frequency 155.37 MHz is related to Conventional Personality – 47. Under Conventional – Conventional Personality – Conventional Personality n ( the configured channel) – Rx Options:



Under Conventional – Conventional Personality – Conventional Personality n (the configured channel) – Tx Options:



## APPENDIX B

### FILTER DESIGN AND COEFFICIENTS

#### Nyquist Raised Cosine Filter

For the design of the Nyquist Raised Cosine Filter, or smoothing filter, the Filter Design Toolbox in MATLAB was used. This resulted in the following code to determine the filter coefficients:

```
%% Raised Cosine Filter
close all;

% All frequency values are in Hz.
Fs = 48000; % Sampling Frequency

N = 1002; % Order
Fc = 2400; % Cutoff Frequency
TM = 'Rolloff'; % Transition Mode
R = 0.2; % Rolloff
DT = 'Normal'; % Design Type

% Create the window vector for the design algorithm.
win = barthannwin(N+1);

% Calculate the coefficients using the FIRRCOS function.
%  $y(n) = b(1)*x(n) + b(2)*x(n-1) + \dots + b(nb+1)*x(n-nb)$ 

b = firrcos(N, Fc/(Fs/2), R, 2, TM, DT, [], win);
b_smoothing = b((502-60):(502+60));

Hd = dfilt.dfir(b_smoothing);

[H_smoothing,w_smoothing] = freqz(Hd);
```

The resulting filter coefficients are as follows:

Table B.1 - Normalized Smoothing Filter Coefficients

| Normalized Coefficient Values |        |        |        |        |        |        |        |        |        |
|-------------------------------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| 0                             | -0.003 | -0.006 | -0.009 | -0.012 | -0.014 | -0.014 | -0.013 | -0.01  | -0.006 |
| 0                             | 0.007  | 0.014  | 0.02   | 0.026  | 0.029  | 0.029  | 0.027  | 0.021  | 0.012  |
| 0                             | -0.013 | -0.027 | -0.039 | -0.049 | -0.054 | -0.055 | -0.049 | -0.038 | -0.021 |
| 0                             | 0.024  | 0.048  | 0.071  | 0.088  | 0.098  | 0.099  | 0.09   | 0.07   | 0.039  |
| 0                             | -0.045 | -0.091 | -0.134 | -0.17  | -0.193 | -0.199 | -0.184 | -0.147 | -0.085 |
| 0                             | 0.105  | 0.227  | 0.36   | 0.496  | 0.629  | 0.751  | 0.854  | 0.933  | 0.983  |
| 1                             |        |        |        |        |        |        |        |        |        |
| 0.983                         | 0.933  | 0.854  | 0.751  | 0.629  | 0.496  | 0.36   | 0.227  | 0.105  | 0      |
| -0.085                        | -0.147 | -0.184 | -0.199 | -0.193 | -0.17  | -0.134 | -0.091 | -0.045 | 0      |
| 0.039                         | 0.07   | 0.09   | 0.099  | 0.098  | 0.088  | 0.071  | 0.048  | 0.024  | 0      |
| -0.021                        | -0.038 | -0.049 | -0.055 | -0.054 | -0.049 | -0.039 | -0.027 | -0.013 | 0      |
| 0.012                         | 0.021  | 0.027  | 0.029  | 0.029  | 0.026  | 0.02   | 0.014  | 0.007  | 0      |
| -0.006                        | -0.01  | -0.013 | -0.014 | -0.014 | -0.012 | -0.009 | -0.006 | -0.003 | 0      |

Note that the upper left coefficient represents coefficient: -60 and that the lower right coefficient represents the coefficient: +60. The value of 1 occurs at zero.

The normalized coefficient values are plotted in the following figure.

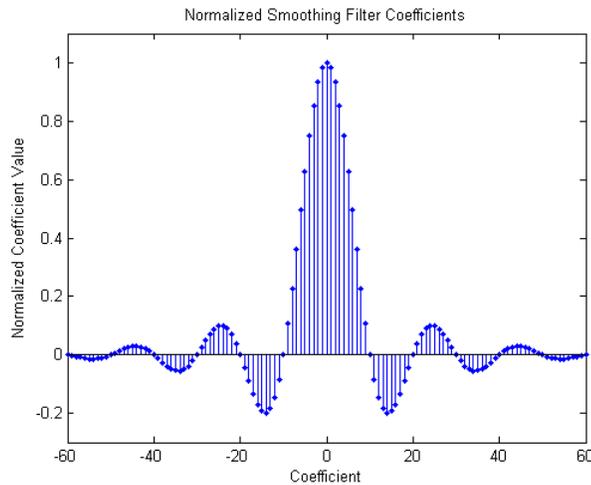


Figure B.1 - Normalized Smoothing Filter Coefficients

### Pre-Emphasis Filter

For the design of the Pre-Emphasis Filter, or shaping filter, the Filter Design Toolbox in MATLAB was used. This resulted in the following code to determine the filter coefficients:

```

% FIR constrained equiripple Inverse Sinc Lowpass filter designed using
% the FIRCEQRIP function.

% All frequency values are in Hz.
Fs = 48000; % Sampling Frequency

N      = 38;      % Order
Fc     = 6800;   % Cutoff Frequency
slope  = 0;      % Stopband Slope
isincfactor = 4; % Inverse Sinc Frequency Factor
isincpower = 1.7; % Inverse Sinc Power
Dstop  = 0.08;   % Stopband Attenuation
Dpass  = 1e-006; % Passband Ripple

% Calculate the coefficients using the FIRCEQRIP function.
b_pe = firceqrrip(N, Fc/(Fs/2), [Dpass, Dstop], 'slope', slope, 'invsinc', ...
    [isincfactor isincpower]);

[H_designed_pe,F_designed_pe] = freqz(b_pe,1,[],Fs);

```

The resulting filter coefficients are:

Table B.2 - Shaping Filter Coefficient Values

| Coefficient Values |        |         |         |         |         |         |        |         |        |
|--------------------|--------|---------|---------|---------|---------|---------|--------|---------|--------|
| -0.018             | 0.0347 | 0.0164  | -0.0064 | -0.0344 | -0.0522 | -0.0398 | 0.0099 | 0.0798  | 0.1311 |
| 0.121              | 0.0322 | -0.113  | -0.2499 | -0.3007 | -0.2137 | 0.0043  | 0.2825 | 0.514   |        |
| 0.604              |        |         |         |         |         |         |        |         |        |
| 0.514              | 0.2825 | 0.0043  | -0.2137 | -0.3007 | -0.2499 | -0.113  | 0.0322 | 0.1214  | 0.1311 |
| 0.080              | 0.0099 | -0.0398 | -0.0522 | -0.0344 | -0.0064 | 0.0164  | 0.0347 | -0.0179 |        |

Note that the upper left coefficient represents coefficient: -19 and that the lower right coefficient represents the coefficient: +19. The value of 0.604 occurs at zero. The coefficient values are plotted in the following figure.

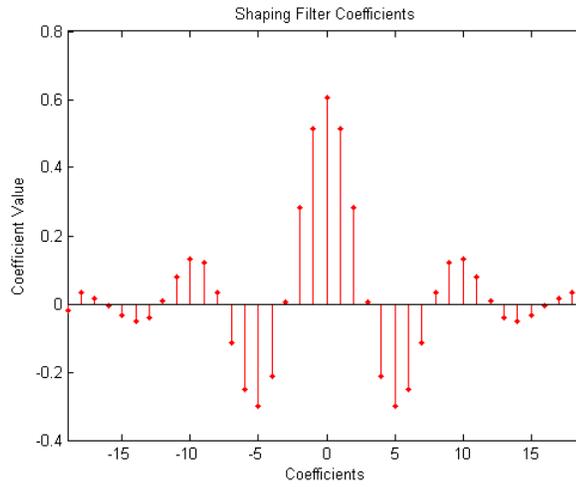


Figure B.2 - Shaping Filter Coefficients

## APPENDIX C

### IP REFLECTOR IN PROJECT54

The purpose of the IP Reflector, which is distributed as part of the Project54 application, is to be able to create a virtual subnet on the server computer, and then to capture all IP traffic sent to that subnet by standard applications running on the server. This provides a consistent method for implementing IP data communications over specialized devices (radio modems) for which standard Windows network adapter drivers are not available. The IP Reflector consists of two components (Figure C.1): a Windows 2000/XP network miniport driver which runs at the kernel level, and a companion Windows service which runs at the Operating System (OS) application level. By splitting the functionality into two pieces, it was possible to minimize the functionality of the kernel level module and to place most of the functionality in the application level module. This both facilitates debugging and allows the functional code to call on any desired Windows OS libraries and services, most of which cannot be accessed by kernel level modules.

On its upper edge, the IP Reflector network miniport driver implements all of the kernel level interfaces required for an Ethernet style network adapter driver. When the driver is installed, it is configured as the connection to an IP subnet (10.\*.\* in Figure C.1). As a result, all IP packets sent by normal applications on the host computer to a 10.\*.\* address are delivered by the OS network stack to

the IP Reflector driver. These outgoing IP packets are encapsulated into outgoing Ethernet frames by the OS network stack before delivery to the driver. In addition to handling IP traffic, the driver must reply appropriately to standard Ethernet ARP requests from the network stack.

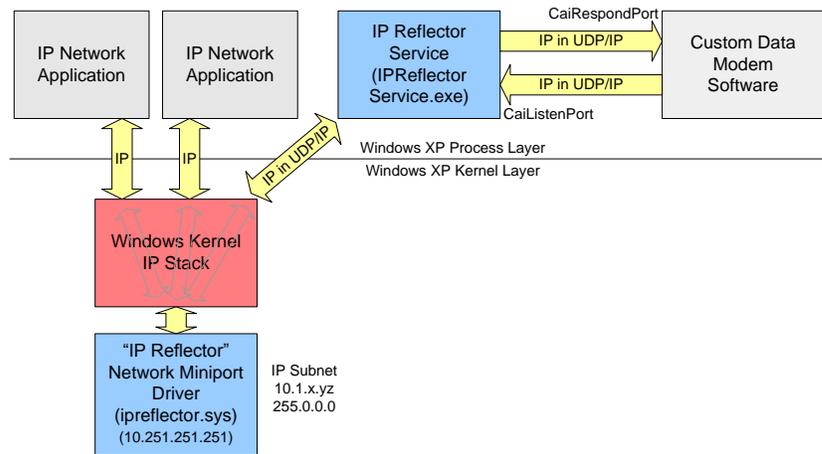


Figure C.1 - The IP Reflector Architecture

When the driver receives an outgoing Ethernet frame from the network stack, it strips off the Ethernet framing to obtain the complete IP packet (See Figure C.2).

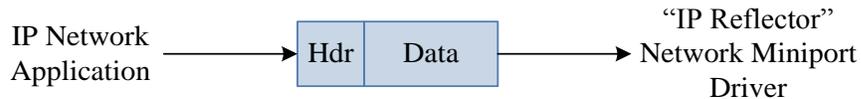


Figure C.2 - IP Packet from IP Network Application to "IP Reflector" Driver

It then packages the complete outgoing IP packet as the payload of a new IP packet (See Figure C.3) addressed to a specific port on the server computer used by the IP Reflector service.



Figure C.3 - IP Packet from "IP Reflector" Driver to IP Reflector Service

Next, it builds an incoming Ethernet frame which contains the new IP packet as its payload (IP encapsulated in UDP/IP encapsulated in Ethernet). This incoming Ethernet frame is then handed back to the OS network stack. The network stack strips off the Ethernet framing and directs the IP packet to the IP Reflector service. The payload of the IP packet received by the service contains the complete original outgoing IP packet (header and data). Finally, the service sends an IP packet, see Figure C.4, to the IP address and port (assigned via the registry) of the program which is actually going to deliver the packet (the Custom Data Modem Software in Figure C.1).

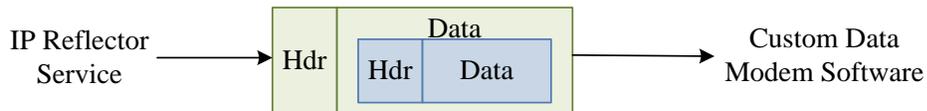


Figure C.4 - IP Packet from IP Reflector Service to Custom Data Modem Software

Note that the IP Reflector driver and service must reside on the same server computer, but the modem software could reside on the same or on a different host computer since the communication between the service and the modem software is based on the IP standard. The custom modem software must then send off the packet appropriately (out through special hardware, for example). In the case of the P25 Base Station design, the Custom Data Modem Software is the TxRx application. The application listens on a specific IP Address and Port which is set in the registry. This is the same address and port to which the IP reflector service transmits messages.

## APPENDIX D

### VSA SETTINGS

This section describes the settings of the Agilent 89441A Vector Signal Analyzer for various measurements.

#### Frequency Spectrum Measurement (Figure III.6)

|                    |              |
|--------------------|--------------|
| Measurement Data   |              |
| Spectrum           | ch1          |
| Data Format        |              |
| Magnitude          | Linear       |
| Frequency          |              |
| Center             | 155.37 MHz   |
| Span               | 5 KHz        |
| Instrument Mode    | Demodulation |
| Demod Type         | Analog       |
| Demodulation Setup |              |
| Ch1 Result         | FM           |

#### APCO 25 Demodulation (Figure III.11 and Figure III.12):

|                               |              |
|-------------------------------|--------------|
| Instrument Mode               | Demodulation |
| Demod Type                    | Digital      |
| Standard Setups               | APCO 25      |
| Frequency                     |              |
| Center                        | 155.37 MHz   |
| Active Trace (Time Measure)   | A            |
| Active Trace (Symbols/Errors) | D            |

## APPENDIX E

### REGISTRY PARAMETERS

Three of the applications which are used in this thesis, acquire parameter information from the registry. All of the applications have registry keys under “HKEY\_LOCAL\_MACHINE\Software\Catlab\Project54\”. The first application is the P25Messenger. Its registry parameters are as follows:

```
{P25Messenger}
  <Network>
    [ReceiveIp]      0.0.0.0
    [ReceivePort]   6540
    [SendIp]        10.1.0.2
    [SendPort]      6540

  <Parameters>
    [Receive]       Enabled or Disabled
    [Transmit]      Enabled or Disabled
```

The settings under the Network key are the IP address for the transmission and reception of IP Packets. If the [ReceiveIp] string value is set to 0.0.0.0, as shown above, the application will listen to any packet. The settings under the Parameters key are for whether the application is set for Receive mode or for Transmit mode.

The next application which uses the registry is the IP Reflector. Its registry parameters are as follows:

```

{Project25}
  <Network>
    [RcpIp]           10.1.254.254
    [RcpPort]         469
    [ReflectorListenPort] 5454
    [ReflectorRespondPort] 10.251.251.251
    [CaiListenPort]   5456
    [CaiRespondIp]    127.0.0.1
    [CaiRespondPort]  5457

```

These settings cause the IP Reflector to transfer all of the IP Packets it receives to the IP address 127.0.0.1 and port 5457.

The final application to use the registry is the TxRx Application. It's registry parameters are as follows:

```

{P25Server}
  <Network>
    [ReceiveIp]       127.0.0.1
    [ReceivePort]     5457
    [SendIp]          127.0.0.1
    [SendPort]        5457
  <Parameters>
    [Port]            COMX

```

The settings under the Network key tell the TxRx application which port to receive packets from the IP Reflector and which port to send messages to the IP Reflector. Under the Parameters key is the [Port] string value. This value is set to the COM Port which will control the push-to-talk.

## APPENDIX F

### CUSTOM RADIO INTERFACE CABLE

A custom interface cable is used to connect the computer to this radio, see Figure F.1 for the schematic. As discussed in Chapter II, when the push-to-talk signal is high, the radio is not transmitting. When the signal is pulled low, the radio begins to transmit. The resistor and the diode are present because the “high” and “low” levels of the Kenwood radio differ from that of the computer’s RS-232. When pin 4 of the DB-9 is high, it outputs a voltage of +9 V. The resistor limits the current input into the radio, protecting the push-to-talk circuitry. When pin 4 of the DB-9 is pulled low, it outputs a voltage of -9 V. The diode limits the input voltage at the push to talk circuit to -0.7 V. This is close enough to 0V to trigger the push-to-talk of the radio, without damaging the circuitry.

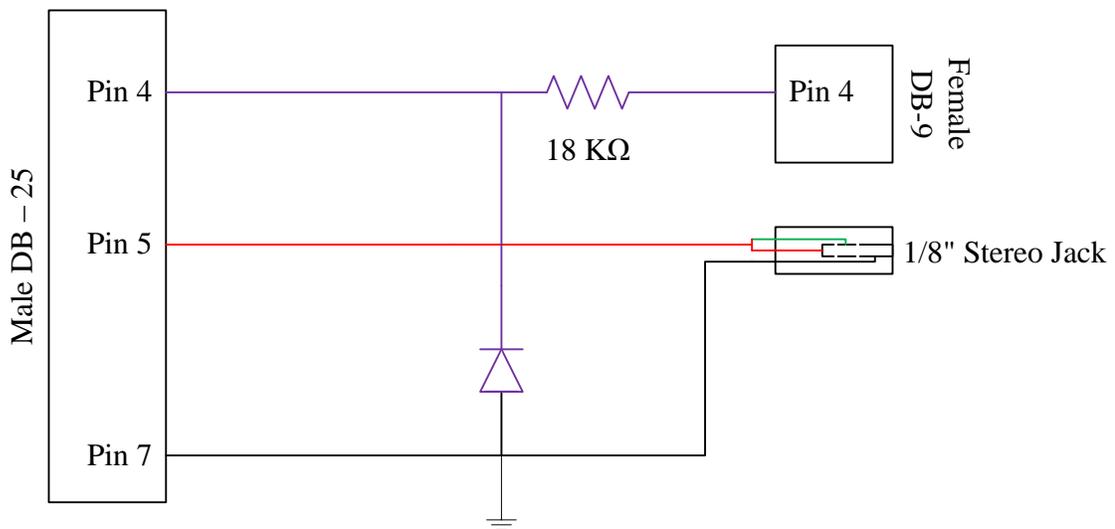


Figure F.1 - Custom Interface Cable